Decidability of invariant validation for paramaterized systems ⋆

Pascal Fontaine and E. Pascal Gribomont

University of Liège (Belgium)
{pfontain,gribomont}@montefiore.ulg.ac.be

Abstract. The control part of many concurrent and distributed programs reduces to a set $\Pi = \{p_1, \ldots, p_n\}$ of symmetric processes containing mainly assignments and tests on Boolean variables. However, the assignments, the guards and the program invariants can be Π-quantified, so the corresponding verification conditions also involve Π-quantifications. We propose a systematic procedure allowing the elimination of such quantifications for a large class of program invariants. At the core of this procedure is a variant of the Herbrand Theorem for many-sorted first-order logic with equality.

1 Introduction

At the heart of concurrent software are control-intensive concurrent algorithms, which solve a large class of problems, including mutual exclusion, termination detection, reliable communication through unreliable channels, synchronous communication through asynchronous channels, fault tolerance, leader election, Byzantine agreement, concurrent reading and writing, and so on. (See e.g. [8, 25] for many examples, with comments and formal or informal proofs). Many of those systems are composed of a parameterized number of identical processes or nearly identical processes[1]. Most variables are Booleans or arrays of Booleans, and operations on the remaining variables are elementary. The verification of such parameterized concurrent systems is the subject of many recent papers [1, 4, 7, 11, 17, 23, 24, 31].

Requirements of such algorithms usually fall in safety properties ("something bad never happens") and liveness properties ("something good eventually happens"). It is often possible to view a liveness property as the conjunction of a safety property and a fairness hypothesis ("progress is made") so, in practice, the verification of safety properties is the main part of formal methods and tools. The classical invariant method allows to reduce the verification of safety properties to the validity problem for first-order logic. It could happen that the

⋆ This work was funded by a grant of the "Communauté française de Belgique - Direction de la recherche scientifique - Actions de recherche concertées"

[1] for example a process can compare its identifier with the identifier of another process. This somewhat breaks symmetry.

formula to be proved belongs to a well known decidable class (for instance, Presburger arithmetic), but this is rarely the case because Boolean arrays (modeled by uninterpreted predicates) are often used in these algorithms, together with interpreted predicates[2].

Quantifier-free first-order logic satisfiability checking is decidable for a very wide range of formulas with non-interpreted and interpreted predicates and functions. Thus decidability is often reached through quantifier elimination. We introduce here a simple quantifier elimination method for a large class of verification conditions. It is based on a many-sorted logic with equality variant of the Herbrand Theorem which allows to have some kind of finite model property [12] even when some functions (interpreted or not) and interpreted predicates are used in formulas. We then give criteria for verification conditions to benefit from this property. Those criteria allow to eliminate quantifiers in the proof by invariant of many reactive algorithms, and particularly for parameterized algorithms, leading to a powerful invariant validation procedure. It allows to reduce the invariant validation for a system with a parameterized number of processes to the invariant validation for a system with a known number of processes n_0 (Theorem 2). Our method can be seen as an extension of the invariant validation procedure presented in [3]: our approach does not restrict the use of functions and predicates to unary ones, and is not restricted to bounded variables.

Our implementation has given good results on several algorithms; in particular, it has been successful in proving all verification conditions for a parameterized railroad crossing system [21] used as benchmark for STeP, whereas STeP itself requires interactive verification for some of them [6].

We first present our variant of the Herbrand Theorem. Next, this variant is used to eliminate the quantifiers in verification conditions from invariant validation of parameterized systems. Last, two examples are presented.

2 Herbrand on many-sorted logic

In this section, Theorem 1 and its context is introduced. This theorem will be used to eliminate quantifiers in verification conditions, which will lead to Theorem 2.

A many-sorted first-order language (a more complete introduction to many-sorted logic can be found in [13]) is a tuple $\mathcal{L} = \langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{P}, r, d \rangle$ such that $\mathcal{T}$ is a finite set of sorts (or types), $\mathcal{V}$ is the (finite) union of disjoint finite sets $\mathcal{V}_\tau$ of variables of sort τ, $\mathcal{F}$ and $\mathcal{P}$ are sets of function and predicate symbols, r $(\mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{N})$ assigns an arity to each function and predicate symbol, and d $(\mathcal{F} \cup \mathcal{P} \rightarrow \mathcal{T}^\star)$ assigns a sort in $\mathcal{T}^{r(f)+1}$ to each function symbol $f \in \mathcal{F}$ and a sort in $\mathcal{T}^{r(p)}$ to each predicate symbol $p \in \mathcal{P}$. Nullary predicates are propositions, and nullary functions are constants.

The sets of τ-terms on language $\mathcal{L}$ contain all variables in $\mathcal{V}_\tau$, and for every function symbol $f \in \mathcal{F}$ of sort $\langle \tau_1, \ldots \tau_n, \tau \rangle$, $f(t_1, \ldots t_n)$ is a τ-term if $t_1, \ldots t_n$ are $\tau_1, \ldots \tau_n$-terms respectively. $Sort(t) = \tau$ if t is a τ-term.

[2] Presburger with (unary) uninterpreted predicates is undecidable [20].

An atomic formula is either $t = t'$ where t and t' are terms of the same sort, or a predicate symbol applied to arguments of appropriate sorts. Formulas are built (as usual) from atomic formulas, connectors ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\equiv$), and quantifiers ($\forall$, $\exists$). The set of all variables used in formula Φ is noted $Vars(\Phi)$, and $Free(\Phi)$ is the set of all free variables in Φ. A formula Φ is closed if $Free(\Phi) = \emptyset$. A formula is τ-universally quantified if it is of the form $\forall x \, \Psi$ with x a variable of type τ.

A formula is in prenex form if it is of the form $Q_1 x_1 \dots Q_n x_n (\Phi)$ where $Q_1, \dots Q_n \in \{\exists, \forall\}$, $x_1, \dots x_n \in \mathcal{V}$, and Φ is quantifier-free. A formula is in Skolem form if it is in prenex form without existential quantifier.

A (normal) interpretation of a formula on a many-sorted first-order language $\mathcal{L} = \langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{P}, r, d \rangle$ is a pair $\mathcal{I} = \langle D, I \rangle$ where

- D assigns a non-empty domain D_τ (set) to each type $\tau \in \mathcal{T}$. Those sets are not necessarily disjoint;
- I assigns an element in D_τ to each variable of sort τ;
- I assigns a function $D_{\tau_1} \times \dots D_{\tau_n} \longrightarrow D_\tau$ to each function symbol $f \in \mathcal{F}$ of sort $\langle \tau_1, \dots \tau_n, \tau \rangle$;
- I assigns a function $D_{\tau_1} \times \dots D_{\tau_n} \longrightarrow \{\top, \bot\}$ to each predicate symbol $p \in \mathcal{P}$ of sort $\langle \tau_1, \dots \tau_n \rangle$;
- the identity is assigned to the equality sign ($=$).

$\mathcal{I}$ assigns a value in D_τ to every τ-term t. This value is noted $\mathcal{I}[t]$. Similarly, interpretation $\mathcal{I}$ assigns a value in $\{\top, \bot\}$ to every formula Φ, which is noted $\mathcal{I}[\Phi]$. An interpretation $\mathcal{I}$ is a model for formula Φ if $\mathcal{I}[\Phi] = \top$. A formula is satisfiable if there exists a model for it.

Given an interpretation $\mathcal{I}$, the congruence $\mathcal{C}_{\mathcal{I},=} = \{(t_i, t'_i) \mid \mathcal{I}[t_i] = \mathcal{I}[t'_i]\}$ is a reflexive, symmetric and transitive relation on the set of terms of language $\mathcal{L}$. This relation is important for the proof of the following theorem.

Theorem 1. *Given*

- *a closed formula S in* Skolem form *on the language $\mathcal{L} = \langle \mathcal{T}, \mathcal{V}, \mathcal{F}, \mathcal{P}, r, d \rangle$;*
- *$\tau \in \mathcal{T}$ such that there is no function symbol $f \in \mathcal{F}$ of sort $\langle \tau_1, \dots \tau_n, \tau \rangle$ with $n > 0$, $\tau_1, \dots \tau_n \in \mathcal{T}$;*

the set H_τ is the set of constant symbols of sort τ ($H_\tau = \{c \in \mathcal{F} \mid d(c) = \tau\}$). If $\{c \in \mathcal{F} \mid d(c) = \tau\} = \emptyset$, then $H_\tau = \{a\}$, where a is an arbitrary new constant symbol such that $a \notin \mathcal{F}$ and $a \notin \mathcal{V}$.

For every model $\mathcal{I} = \langle D, I \rangle$ of S, there is a model $\mathcal{I}' = \langle D', I' \rangle$ such that

- *D'_τ is the quotient of the set H_τ by congruence $\mathcal{C}_{\mathcal{I},=}$;*
- *$D'_{\tau'} = D_{\tau'}$ for every $\tau' \neq \tau$;*
- *$\mathcal{I}'[f] = \mathcal{I}[f]$ for every function symbol $f \in \mathcal{F}$ of sort $\langle \tau_1, \dots \tau_n, \tau' \rangle$ such that $\tau_1 \neq \tau, \dots \tau_n \neq \tau, \tau' \neq \tau$ ($n \geq 0$);*
- *$\mathcal{I}'[p] = \mathcal{I}[p]$ for every function symbol $p \in \mathcal{P}$ of sort $\langle \tau_1, \dots \tau_n \rangle$ such that $\tau_1 \neq \tau, \dots \tau_n \neq \tau$ ($n \geq 0$).*

Proof. Interpretation $\mathcal{I}'$ is built from $\mathcal{I}$:

– for every constant symbol c of sort τ in $\mathcal{F}$, $I'[c]$ is the class of c in D_τ;
– for every function symbol $f \in \mathcal{F}$ of sort $\langle \tau_1, \dots \tau_n, \tau' \rangle$ $(n > 0)$, and every $d'_1 \in D'_{\tau_1}, \dots d'_n \in D'_{\tau_n}$, $I'[f](d'_1, \dots d'_n) = I[f](d_1, \dots d_n)$ where $d_i = d'_i$ if $\tau_i \neq \tau$. If $\tau_i = \tau$, $d_i = I(d''_i)$ where d''_i is any element of the class $d'_i \in D_\tau$;
– for every predicate symbol $p \in \mathcal{P}$ of sort $\langle \tau_1, \dots \tau_n \rangle$, and every elements $d'_1 \in D'_{\tau_1}, \dots d'_n \in D'_{\tau_n}$, $I'[p](d'_1, \dots d'_n) = I[p](d_1, \dots d_n)$ where $d_i = d'_i$ if $\tau_i \neq \tau$. If $\tau_i = \tau$, $d_i = I(d''_i)$ where d''_i is any element of the class $d'_i \in D_\tau$.

It remains to show that $\mathcal{I}'$ is a model of S. Let us first introduce a notation: given an interpretation $\mathcal{J} = \langle D, J \rangle$, the interpretation $\mathcal{J}_{x_1/d_1, \dots x_n/d_n} = \langle D, J' \rangle$ (where $x_1, \dots x_n$ are variables) is such that $J'[x_i] = d_i$ for every $x_i \in \{x_1, \dots x_n\}$ and $J'[t] = J[t]$ if $t \notin \{x_1, \dots x_n\}$

Formula S is of the form $\forall x_1 \dots \forall x_n (\Phi)$. Thus for all elements $d'_1, \dots d'_n$ such that d'_i belongs to $D'_{\tau'}$ if x_i is a variable of sort τ', the following equality hold:

$$\mathcal{I}'_{x_1/d'_1, \dots x_n/d'_n}[\Phi] = \mathcal{I}_{x_1/d_1, \dots x_n/d_n}[\Phi]$$

with $d_i = \mathcal{I}[d''_i]$ where d''_i is any element of the class $d'_i \in D'_\tau$ if x_i is of sort τ, $d_i = d'_i$ otherwise.

Interpretation $\mathcal{I}$ is a model of formula S, that means $\mathcal{I}_{x_1/d_1, \dots x_n/d_n}[\Phi] = \top$ for all elements $d_1, \dots d_n$ where d_i belongs to $D_{\tau'}$ if x_i is a variable of sort τ'. It follows that $\mathcal{I}'_{x_1/d'_1, \dots x_n/d'_n}[\Phi] = \top$ for all elements $d'_1, \dots d'_n$ such that d'_i belongs to $D'_{\tau'}$ if x_i is a variable of sort τ'. So $\mathcal{I}'$ is a model of S. $\square$

This theorem is not exactly an extension of the Herbrand theorem to many-sorted first-order logic. It is stronger than the Herbrand theorem (see for example [14] for the standard Herbrand theorem, or [16] for a version with equality) in the sense that the domain does not necessarily become infinite in the presence of functions. On the other hand, its restriction to one-sorted first-order logic gives back the Herbrand theorem, but restricted to the finite Herbrand universe case. Nevertheless this case is the most interesting one: having a finite domain means that quantifier elimination is possible. Consider the simple (unsatisfiable) formula

$$\forall i \forall j \, [f(i) > g(j)] \; \wedge \; g(a) = 3 \; \wedge \; \exists i \, [f(i) < 4] \tag{1}$$

where "$<$" and "$>$" are the usual order predicates on $\mathbb{N} \times \mathbb{N}$. Variables i and j and constants a and b are of sort $\tau \neq \mathbb{N}$ whereas f and g are functions from τ to $\mathbb{N}$. In this context, the preceding theorem states that formula (1) is satisfiable if and only if formula

$$\begin{aligned} & f(a) > g(a) \; \wedge \; f(a) > g(b) \; \wedge \; f(b) > g(a) \; \wedge \; f(b) > g(b) \\ & \quad \wedge g(a) = 3 \; \wedge \; f(b) < 4 \end{aligned}$$

is. This last formula belongs to the decidable class of quantifier-free first-order logic with linear arithmetics on $\mathbb{N}$ and uninterpreted function symbols.

Corollary 1. *A τ-universally quantified formula $\forall x \, \Phi(x)$ verifying the conditions of Theorem 1 is satisfiable if and only if the finite conjunction $\bigwedge_{c \in H_\tau} \Phi(c)$ is.*

3 Interpreted predicates and functions

A formula containing interpreted predicates and functions is satisfiable if and only if it has a model in a restricted subset of all interpretations, that is the set where interpretations associate a fixed domain to given sorts and a fixed meaning to those interpreted predicates and functions. In Theorem 1, both interpretations $\mathcal{I}$ and $\mathcal{I}'$ associate the same domain to every sort but τ, and give the same meaning to every predicate and function, provided none of their arguments is of sort τ. In other words, Theorem 1 is compatible with the use of interpreted predicates and functions provided none of their arguments is of sort τ. For instance, in the preceding example (i, j and a are of sort τ) the arguments of the order predicates ($f(i)$, $g(j)$, ...) are not of the sort τ. Using Theorem 1, interpretation $\mathcal{I}$ and $\mathcal{I}'$ are such that $\mathcal{I}[<] = \mathcal{I}'[<]$ and $\mathcal{I}[>] = \mathcal{I}'[>]$. And this allows to eliminate the quantifiers on the sort τ in presence of interpreted predicates with no argument of sort τ.

But it is also possible to use order predicates on the sort of quantified variables. Let φ be a formula with order predicates ("$\leq$", ...) on sort τ, and ψ be the conjunction of the axioms of total order theory,

$$\begin{aligned}\psi = \quad & \forall x\,(x \leq x) \\ & \wedge\ \forall x \forall y\,((x \leq y \wedge y \leq x) \Rightarrow x = y) \\ & \wedge\ \forall x \forall y \forall z\,((x \leq y \wedge y \leq z) \Rightarrow x \leq z) \\ & \wedge\ \forall x \forall y\,(x \leq y \vee y \leq x)\end{aligned}$$

with variables x, y, z of sort τ. An interpretation is a model of $\psi \wedge \varphi$ if and only if it is a model of φ interpreting "$\leq$", ... as the usual order predicates on D_τ. Putting $\psi \wedge \varphi$ in Skolem form does not introduce new Skolem functions. The conditions of Theorem 1 are met for $\psi \wedge \varphi$ if they are met for φ. Theorem 1 can be applied also if some comparisons are made between terms of the sort of quantified variables[3].

4 Quantifier elimination in invariant validation

In order to verify that the assertion H is an invariant of the transition system $\mathcal{S}$, one has to validate the Hoare triple $\{H\}\sigma\{H\}$ for each transition[4] $\sigma \in \mathcal{S}$. This is first reduced to first-order logic proving, using Dijkstra [9] weakest precondition (wp) operator: Hoare tripe $\{H\}\sigma\{H\}$ is valid if and only if formula $H \Rightarrow wp[\sigma; H]$ can be proved. Weakest precondition calculus is easy, provided

[3] As in [3], "+1" and "$\oplus$1" functions can sometimes be eliminated without introducing new Skolem functions, by noticing that $h = i + 1 \longleftrightarrow i < h \wedge \forall j\,(j \leq i \vee h \leq j)$ and $h = i \oplus 1 \longleftrightarrow [i < h \wedge \forall j\,(j \leq i \vee h \leq j)] \vee [h < i \wedge \forall j\,(h \leq j \vee j \leq i)]$.

[4] An example of transition is $(s_0[p]s_0[q], C \longrightarrow A, s_1[p]s_1[q])$ which allows the processes p and q to go from control point s_0 to control point s_1, executing the statements in A. The system transition can be executed from a state where formula C (the guard) is fulfilled.

transitions do not contain full loops in their statement part. The weakest precondition module in CAVEAT accepts assignments, conditional statements, sequences of statements, and some kind of quantified assignments. This is enough to model reactive algorithms from coarse to fine-grained versions.

In general, the invariant is a conjunction ($H = \bigwedge_{k \in K} h_k$) of relatively small assertions h_k. In parameterized systems, these assertions are often quantified over the (parameterized) set of processes. In order to avoid the appearance of Skolem functions when verification conditions are put in Skolem form, an assumption is made about these quantified assertions: they can be put both in prenex form $\exists^\star\forall^\star$ (called *hypothesis form* in the following, because this will be the allowed form in the antecedent of formulas of the form $A \Rightarrow B$) *and* in prenex form $\forall^\star\exists^\star$ (called *conclusion form* in the following, because this will be the allowed form in the conclusion of formulas of the form $A \Rightarrow B$). In practice, two particular cases of such formulas are met frequently:

- formulas in prenex form containing one type of quantifier;
- formulas containing only monadic predicates (and no equality)[5].

There is also an assumption for guards : guards must be formulas in hypothesis form. Guards met in practice fulfill this assumption as they are quantifier-free formulas or singly quantified formulas.

Taking the preceding conditions on quantifiers into account, proving formula $H \Rightarrow wp[\sigma; H]$ (with $H = \bigwedge_{k \in K} h_k$) reduces to prove a set of formulas (called verification conditions) of the form

$$(h_1 \wedge \ldots h_k \wedge G) \Rightarrow C_j$$

where G is the guard of σ. All formulas $h_1 \ldots h_k, G$ are in hypothesis form. There is one verification condition for each h_k ($k \in K$). Formula C_k comes from hypothesis h_k: $C_k \equiv wp\,[A; h_k]$, where A is the statement part of σ. C_k can be put in conclusion form: indeed, h_k can be put in conclusion form, and the weakest precondition operator does not modify the quantifier structure of a formula, in the language accepted by CAVEAT.

The last requirement is about functions: we require that no function used in the invariant H, or in the transition system $\mathcal{S}$ has the process set as domain. This may seem rather restrictive, but as reactive algorithms mainly use Boolean arrays (modeled by predicates, not functions), this requirement remains acceptable in practice.

Under those conditions, Theorem 1 can be used to eliminate the quantifiers:

Theorem 2. *If H is a conjunctive formula, and Σ is a transition system with a parameterized number n of processes, where*

- *all quantified variables in H and in the guard of the transitions of Σ range over the set of processes;*

[5] Indeed every monadic formula is logically equivalent to a Boolean combination of Skolem forms with one quantifier. So every monadic formula can be put in both hypothesis and conclusion forms.

- *every conjunct in H can be put both in hypothesis form ($\exists^\star\forall^\star$) and in conclusion form ($\forall^\star\exists^\star$);*
- *every transition guard can be put in hypothesis form;*
- *no interpreted predicate other than equality and order is used on the set of processes, neither in H nor in Σ;*
- *no function has the process set as domain, neither in H nor in Σ;*

then H is an invariant of Σ if and only if H is an invariant of the system Σ' with at most n_0 processes, where n_0 is the sum of

- *the number of existential quantifiers in H when put in hypothesis form;*
- *the maximum number of existential quantifiers in guards of transitions in Σ;*
- *the maximum number of universal quantifiers in the conjuncts of H, when put in conclusion form;*
- *the number of constants in H;*
- *the maximum number of processes taking part in a transition[6].*

Proof. Indeed from the theorem conditions, every verification condition is of the form

$$(h_1 \wedge \dots h_k \wedge G) \Rightarrow C$$

where formulas $h_1, \dots h_k, G$ are in hypothesis form, and C is in conclusion form. When put prenex form, this formula is of the form

$$\forall x_1 \dots \forall x_p \exists y_1 \dots \exists y_q\, \varphi(x_1, \dots x_p, y_1, \dots y_q), \tag{2}$$

where p is the number of existential quantifiers in $h_1 \wedge \dots h_k \wedge G$ plus the number of universal quantifiers in C. Otherwise stated, p cannot exceed the sum of

- the number of existential quantifiers in H ($h_1 \wedge \dots h_k$) when put in hypothesis form;
- the maximum number of existential quantifiers in guards (G) of transitions in Σ;
- the maximum number of universal quantifiers in the conjuncts (h_k from which C is computed) of H, when put in conclusion form.

Formula (2) is provable if and only if formula

$$\exists x_1 \dots \exists x_p \forall y_1 \dots \forall y_q\, \neg\varphi(x_1, \dots x_p, y_1, \dots y_q) \tag{3}$$

is unsatisfiable or, using Skolemization, if and only if formula

$$\forall y_1 \dots \forall y_q\, \neg\varphi(a_1, \dots a_p, y_1, \dots y_q) \tag{4}$$

is unsatisfiable, where $a_1, \dots a_p$ are Skolem constants, i.e. constants which do not appear in $\varphi(x_1, \dots x_p, y_1, \dots y_q)$. Using Theorem 1, formula (4) is satisfiable if and only if there is a model with a finite process set, which contains all process constants in $\varphi(a_1, \dots a_p, y_1, \dots y_q)$, including $a_1, \dots a_p$. So n_0 is the sum of

[6] usually at most two.

– p;
– the number of constants coming from H in φ;
– the maximum number of constants coming from the transitions through G and C, which is the maximum number of processes involved at the same time in a transition.

$\square$

Comment. The satisfiability problem for the Schönfinkel-Bernays class, that is, the class of function-free first-order formulas of the form

$$\exists x_1 \dots \exists x_p \forall y_1 \dots \forall y_q \, \varphi(x_1, \dots x_p, y_1, \dots y_q),$$

has first been shown to be decidable by Bernays and Schönfinkel without equality [5] and by Ramsey with equality [29]. Theorem 1 extends this decidable class to allow the use of some functions (interpreted or not) and some interpreted predicates.

Corollary 2. *When conditions of Theorem 2 are met, checking if Σ preserves the invariant H is reduced to a quantifier-free first-order logic satisfiability checking problem.*

The quantifier-free satisfiability checking module [15] in CAVEAT is based on a modified version of the Nelson-Oppen algorithm [26, 27]. It accepts linear arithmetic, as well as uninterpreted predicates and functions. When Theorem 2 applies, and when the quantifier-free formulas use only linear arithmetic, and uninterpreted predicates and functions, the invariant validation problem is decidable. This is the case for numerous algorithms. In the next section a simple one is presented.

5 Parameterized Burns algorithm

In this well-known simple example only one type of variable is used. Theorem 1 thus reduces to the Herbrand theorem (with equality, without functions). This simple example allows to clearly exhibit the underlying fact which enables quantifier elimination: a finite Herbrand universe.

Burns algorithm [22], [25, p. 294] guarantees exclusive access to a critical section for a set of n identical processes. Each process p can be in one of six different location states (i.e. $s_0 \dots s_5$). A rule expresses the trivial property that each process is in one and only one state at each time: one and only one variable in $s_0[p]$,..., $s_5[p]$ is true (for each p). A process p being in s_5 (i.e. $s_5[p]$ is true) is in the critical section.

Twelve transitions are possible between the six states:

$$\big(\mathrm{s}_0[p],\ \mathrm{flag}[p]{:=}\mathrm{false},\ \mathrm{s}_1[p]\big)$$
$$\big(\mathrm{s}_1[p], \neg S[p,q] \ \wedge \ q < p \ \wedge \ \mathrm{flag}[q] \ \rightarrow \ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_0[p]\big)$$
$$\big(\mathrm{s}_1[p], \neg S[p,q] \ \wedge \ q < p \ \wedge \ \neg\mathrm{flag}[q] \ \rightarrow \ S[p,q]{:=}\mathrm{true},\ \mathrm{s}_1[p]\big)$$
$$\big(\mathrm{s}_1[p], \forall q\big(q < p \Rightarrow S[p,q]\big) \ \rightarrow \ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_2[p]\big)$$

$$\big(\mathrm{s}_2[p],\ \mathrm{flag}[p]{:=}\mathrm{true},\ \mathrm{s}_3[p]\big)$$
$$\big(\mathrm{s}_3[p], \neg S[p,q]\ \wedge\ q<p\ \wedge\ \mathrm{flag}[q]\ \rightarrow\ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_0[p]\big)$$
$$\big(\mathrm{s}_3[p], \neg S[p,q]\ \wedge\ q<p\ \wedge\ \neg\mathrm{flag}[q]\ \rightarrow\ S[p,q]{:=}\mathrm{true},\ \mathrm{s}_3[p]\big)$$
$$\big(\mathrm{s}_3[p], \forall q\big(q<p \Rightarrow S[p,q]\big)\ \rightarrow\ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_4[p]\big)$$
$$\big(\mathrm{s}_4[p], \neg S[p,q]\ \wedge\ p<q\ \wedge\ \mathrm{flag}[q]\ \rightarrow\ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_4[p]\big)$$
$$\big(\mathrm{s}_4[p], \neg S[p,q]\ \wedge\ p<q\ \wedge\ \neg\mathrm{flag}[q]\ \rightarrow\ S[p,q]{:=}\mathrm{true},\ \mathrm{s}_4[p]\big)$$
$$\big(\mathrm{s}_4[p], \forall q\big(p<q \Rightarrow S[p,q]\big)\ \rightarrow\ \forall q : S[p,q]{:=}\mathrm{false},\ \mathrm{s}_5[p]\big)$$
$$\big(\mathrm{s}_5[p],\ \mathrm{flag}[p]{:=}\mathrm{false},\ \mathrm{s}_0[p]\big)$$

Mutual exclusion is obtained using two waiting rooms (s_3 and s_4). The first one ensures that when a process p has reached s_4, any other process q with $q < p$ and flag$[q]$ = true (trying to get access to critical section, or in the critical section) has gone through transition $s_2 \rightarrow s_3$ *after* p. The second waiting room guarantees that this process q (with $q < p$) will be blocked in s_4 at least until p resets flag$[p]$ to false. Only the highest process (the one with the highest identifier) will thus get access to critical section[7].

The algorithm uses one single-writer shared register per process: flag$[p]$ is set to true by process p when it wants to access to critical section. Each process p also uses a local array variable $S[p]$. This variable is used in three loops (s_1, s_3, s_4). In the loops for process p the value of the flag$[q]$ variable of the other processes q is checked (processes q such that $q < p$ or $q > p$). $S[p]$ is used to keep track of processes already checked and those which still have to be checked. The algorithm makes also extensive use of a total order relation between processes.

Formula $H =_{\mathrm{def}} \forall p\ H_1(p) \wedge \forall p \forall q\,[H_2(p,q) \wedge H_3(p,q)]$, with

$$\begin{aligned}
H_1(p) &=_{\mathrm{def}} \neg\mathrm{flag}[p] \Rightarrow (\mathrm{s}_0[p] \vee \mathrm{s}_1[p] \vee \mathrm{s}_2[p])\\
H_2(p,q) &=_{\mathrm{def}} \mathrm{s}_2[p] \Rightarrow \neg S[p,q]\\
H_3(p,q) &=_{\mathrm{def}} \big[q<p\ \wedge\ \mathrm{flag}[q]\ \wedge\ (\mathrm{s}_5[p] \vee \mathrm{s}_4[p] \vee (\mathrm{s}_3[p] \wedge S[p,q]))\,\big]\\
&\qquad\qquad \Rightarrow \big[\neg\mathrm{s}_5[q] \wedge \neg(\mathrm{s}_4[q] \wedge S[q,p])\big]
\end{aligned}$$

is an invariant. It entails[8] the mutual exclusion property:

$$\forall p \forall q \big[p \neq q \Rightarrow (\neg s_5[p] \vee \neg s_5[q])\big].$$

Every condition is met for Theorem 2 to be used. Indeed:

- no function (at all) is used;
- every guard is in hypothesis form. In fact, every guard is at most once quantified;

[7] Access to critical section will be easier for processes with high identifiers. This algorithm does not guarantee high-level-fairness.

[8] together with the rule which expresses the fact that each process is in one and only one state at a time.

- the invariant is a conjunction of formulas which are in both hypothesis and conclusion form, as they are universally quantified;
- the only interpreted predicates are equality and order; objects compared belong to a finite, but parameterized, domain: the set of processes.

From Theorem 2, if H is an invariant of this algorithm for $n_0 = 4$ *processes* then H will be an invariant of this algorithm *for any number of processes.*

Let's see how this work for a given verification condition: if H is an invariant, it is preserved by every transition, and in particular, by transition $\sigma_{1\to 2}$ from s_1 to s_2. Hoare triple $\{H\}\sigma_{1\to 2}\{H\}$ must be provable, so must be $\{H\}\sigma_{1\to 2}\{\forall p\ H_1(p)\}$, $\{H\}\sigma_{1\to 2}\{\forall p\forall q\ H_2(p,q)\}$ and $\{H\}\sigma_{1\to 2}\{\forall p\forall q\ H_3(p,q)\}$. In particular, from $\{H\}\sigma_{1\to 2}\{\forall p\forall q H_2(p,q)\}$ comes the verification condition

$$\varphi =_{\text{def}} (h_1 \wedge h_2 \wedge h_3 \wedge g_1 \wedge g_2 \wedge l_1 \wedge l_2 \wedge l_3 \wedge l_4 \wedge l_5) \Rightarrow C$$

with

- $h_1 =_{\text{def}} \forall p\ H_1(p)$
- $h_2 =_{\text{def}} \forall p\forall q\ H_2(p,q)$
- $h_3 =_{\text{def}} \forall p\forall q\ H_3(p,q)$
- $g_1 =_{\text{def}} s_1[p]$
- $g_2 =_{\text{def}} \forall q\big[q < p \Rightarrow S[p,q]\big]$
- $l_1 =_{\text{def}} \forall p\big[s_0[p] \Rightarrow \neg(s_1[p] \vee s_2[p] \vee s_3[p] \vee s_4[p] \vee s_5[p])\big]$
- $l_2 =_{\text{def}} \forall p\big[s_1[p] \Rightarrow \neg(s_2[p] \vee s_3[p] \vee s_4[p] \vee s_5[p])\big]$
- $l_3 =_{\text{def}} \forall p\big[s_2[p] \Rightarrow \neg(s_3[p] \vee s_4[p] \vee s_5[p])\big]$
- $l_4 =_{\text{def}} \forall p\big[s_3[p] \Rightarrow \neg(s_4[p] \vee s_5[p])\big]$
- $l_5 =_{\text{def}} \forall p\big[s_4[p] \Rightarrow \neg s_5[p]\big]$
- $C =_{\text{def}} \forall s\forall r\big[(s \neq p \Rightarrow s_2[s]) \Rightarrow \neg(s \neq p\ \wedge\ S[s,r])\big]$

Hypotheses $h_{1,2,3}$ come from the invariant, $g_{1,2}$ from the transition guards[9]. Formulas $l_{1,\ldots 5}$ state that each process is in one and only one state. The conclusion C is the result of applying the weakest precondition operator, i. e.,

$$C \equiv wp\ [\forall q : S[p,q]{:=}\text{false}; \mathrm{s}_1[p] := \text{false}; \mathrm{s}_2[p] := \text{true}; \forall p\forall q\ H_2(p,q)]$$

Every formula from h_1 to l_5 is in hypothesis form, and C is in conclusion form. The Herbrand universe for the negation of this verification condition contains four elements (p, q, and the new constants coming from the Skolemization of C). Every universal quantifier in hypotheses will then give rise to four instances, for a total of 61 hypotheses[10].

Caveat took 5 seconds on a Pentium 1 GHz, to generate and verify 40 verification conditions. This includes the time to verify that the invariant entails the mutual exclusion property, and also that the invariant is made true by initial conditions.

[9] g_1 comes from the origin of the transition. Transition $(l_1, C \longrightarrow A, l_2)$ with origin l_1 and destination l_2 can be written as transition $((C \wedge l_1) \longrightarrow A; l_1 := \text{false}; l_2 := \text{true})$.

[10] each formula h_1, g_2, $l_{1\ldots 5}$ generates four instances, whereas formulas h_2 and h_3 generate 16 instances. The 61st hypothesis is g_1.

6 Generalized Railroad Crossing

The Generalized Railroad Crossing benchmark [21] uses predicates and functions from arithmetic. It gives a general idea of what Theorem 1 allows to deal with.

A controller operates on a gate of a railroad crossing protecting N parallel railroad tracks. The gate must be down whenever a train takes the intersection, so that the intersecting road is closed. Each of the N trains can be in three different regions: in the intersection (I), in the section preceding the intersection (P), or anywhere else (not_here). The array variable "trains" records the position of each train: trains[i] can be one of the three values $I, P, \text{not\_here}$. The gate can be in four states: the value of variable "gate" can be $\text{down}, \text{up}, \text{going\_down}$ or going_up, with obvious meanings. The system should verify the safety property, which expresses the fact that the gate must be down when any of the N trains is passing the intersection:

$$\forall i\,(\text{trains}[i] = I \Rightarrow \text{gate} = \text{down})\,.$$

The gate takes some time to go from the state "up" to "down". This time must not exceed "gateRiseTime". Similarly the time to go from the "down" to the "up" states must not exceed "gateDownTime". Trains getting in P would take a minimum time "minTimeToI" and a maximum time "maxTimeToI" to get to the intersection. It is the controller job to know when to lower the gate, and when to raise it. Initially, the gate is up, and no train is either in the intersection or in the section preceding the intersection.

The system transitions are given on Figure 1. The first three transitions model the position changes of the train i. The two following ones express the controller decision to lower or raise the gate. The next two mean the gate reaches the up or down states. The last one models the time flow.

Only two transition guards are not quantifier-free. But they can easily be put in prenex form with a single quantifier. Functions are used (trains, firstEnter, lastEnter, schedTime, +) but they do not range over the process set. All requirements are thus met for Theorem 2 to be used, as long as the invariants to be checked also verify the requirements about quantifiers.

Figure 2 shows several invariance properties of the system. Together with the safety property, they give an invariant for the system. As the safety property is one conjunct of the invariant, it is trivially entailed by the invariant. In order to validate the invariant, it is necessary to take into account the constraints on constants (Figure 3) as well as the progress axioms[11] (Figure 4). They are supplementary hypotheses to be put in the verification conditions.

In the whole proof, only two properties (or guards) are existentially quantified, properties are at most once quantified, and at most one train take part in a transition. From Theorem 2, if the invariant (which guarantees that the algorithm is safe) is preserved *for four trains*, the algorithm will be safe *for any number of trains*.

[11] For example, progress axiom P_1 states that the train does not stay indefinitely in section P before going in I.

$$
\begin{aligned}
&\big(\mathrm{trains}[i] = \mathrm{not\_here} \;\longrightarrow\; \mathrm{begin}\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathrm{trains}[i] := P;\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathrm{firstEnter}[i] := T + \mathrm{minTimeToI};\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathrm{lastEnter}[i] := T + \mathrm{maxTimeToI};\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathrm{schedTime}[i] := T + \mathrm{conMinI};\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathrm{trainHere}[i] := \mathrm{true}\\
&\qquad\qquad\qquad\qquad\quad \mathrm{end}\big)
\end{aligned}
$$

$$
\big(\mathrm{trains}[i] = P \;\wedge\; T \geq \mathrm{firstEnter}[i] \;\longrightarrow\; \mathrm{trains}[i] := I\big)
$$

$$
\big(\mathrm{trains}[i] = I \;\longrightarrow\; \mathrm{begin\ trains}[i] := \mathrm{not\_here};\; \mathrm{trainHere}[i] := \mathrm{false\ end}\big)
$$

$$
\begin{aligned}
&\big(\ (\mathrm{gate} = \mathrm{up} \;\vee\; \mathrm{gate} = \mathrm{going\_up}) \;\wedge\; \mathrm{gstatus} = \mathrm{up}\\
&\quad \wedge \exists i\,(\mathrm{trainHere}[i] \;\wedge\; \mathrm{schedTime}[i] \leq T + \gamma_{\mathrm{down}} + \beta)\\
&\quad \longrightarrow\; \mathrm{begin}\\
&\qquad\qquad \mathrm{gate} = \mathrm{going\_down};\\
&\qquad\qquad \mathrm{lastDown} := T + \mathrm{gateDownTime};\\
&\qquad\qquad \mathrm{gstatus} := \mathrm{down}\\
&\qquad\quad \mathrm{end}\big)
\end{aligned}
$$

$$
\begin{aligned}
&\big(\ (\mathrm{gate} = \mathrm{down} \;\vee\; \mathrm{gate} = \mathrm{going\_down}) \;\wedge\; \mathrm{gstatus} = \mathrm{down}\\
&\quad \wedge \forall i\,(\mathrm{trainHere}[i] \Rightarrow \mathrm{schedTime}[i] > T + \gamma_{\mathrm{down}} + \gamma_{\mathrm{up}} + \mathrm{carPassingTime})\\
&\quad \longrightarrow\; \mathrm{begin}\\
&\qquad\qquad \mathrm{gate} := \mathrm{going\_up};\\
&\qquad\qquad \mathrm{lastUp} := T + \mathrm{gateRiseTime};\\
&\qquad\qquad \mathrm{gstatus} := \mathrm{up}\\
&\qquad\quad \mathrm{end}\ \big)
\end{aligned}
$$

$$
\big(\mathrm{gate} = \mathrm{going\_up} \;\longrightarrow\; \mathrm{gate} := \mathrm{up}\big)
$$

$$
\big(\mathrm{gate} = \mathrm{going\_down} \;\longrightarrow\; \mathrm{gate} := \mathrm{down}\big)
$$

$$
\big(T := T + \varepsilon\big)
$$

Fig. 1. The transitions modeling the General Railroad Crossing system

$$T_1 =_{\text{def}} \forall i \left(T < \text{firstEnter}[i] \Rightarrow \text{trains}[i] \neq I\right)$$

$$\begin{aligned} T_2 =_{\text{def}} \forall i \big(&\text{trains}[i] = P \Rightarrow \\ &(\text{firstEnter}[i] \leq T + \text{minTimeToI} \ \wedge \ T \leq \text{lastEnter}[i] \\ &\wedge \text{lastEnter}[i] - \text{firstEnter}[i] = \text{maxTimeToI} - \text{minTimeToI})\big) \end{aligned}$$

$$C_1 =_{\text{def}} \text{gstatus} = \text{up} \Rightarrow \forall i \left(\text{trainHere}[i] \Rightarrow T < \text{schedTime}[i] - \gamma_{\text{down}}\right)$$

$$GC_1 =_{\text{def}} \text{gstatus} = \text{down} \equiv (\text{gate} = \text{goingDown} \ \vee \ \text{gate} = \text{down})$$

$$GC_2 =_{\text{def}} \text{gstatus} = \text{down} \Rightarrow \text{lastDown} \leq T + \text{gateDownTime}$$

$$GC_3 =_{\text{def}} \text{gstatus} = \text{up} \Rightarrow \forall i \left(\text{trainHere}[i] \Rightarrow \text{lastDown} < \text{schedTime}[i]\right)$$

$$TC_1 =_{\text{def}} \forall i \left(\text{trainHere}[i] \equiv \text{trains}[i] \neq \text{notHere}\right)$$

$$TC_2 =_{\text{def}} \forall i \left(\text{trainHere}[i] \Rightarrow \text{schedTime}[i] < \text{firstEnter}[i]\right)$$

Fig. 2. Invariance properties

$$AC_1 := \gamma_{\text{down}} < \text{conMinI}$$

$$ACT_1 := \text{conMinI} < \text{minTimeToI}$$

$$AGC_1 := \text{gateDownTime} < \gamma_{\text{down}}$$

$$AGC_2 := \text{gateRiseTime} < \gamma_{\text{up}}$$

Fig. 3. Constraints on constants

$$G_1 =_{\text{def}} \text{gate} = \text{goingDown} \Rightarrow T \leq \text{lastDown}$$

$$G_2 =_{\text{def}} \text{gate} = \text{goingUp} \Rightarrow T \leq \text{lastUp}$$

$$P_1 =_{\text{def}} \forall i \left(\text{trains}[i] = P \Rightarrow T \leq \text{lastEnter}[i]\right)$$

$$P_2 =_{\text{def}} \text{gstatus} = \text{up} \Rightarrow \forall i \left(\text{trainHere}[i] \Rightarrow T < \text{schedTime}[i] - \gamma_{\text{down}}\right)$$

$$\begin{aligned} P_3 =_{\text{def}} \ &\text{gstatus} = \text{down} \Rightarrow \\ &\exists i \left(\text{trainHere}[i] \ \wedge \ \text{schedTime}[i] \leq T + \gamma_{\text{up}} + \text{carPassingTime} + \gamma_{\text{down}}\right) \end{aligned}$$

Fig. 4. Progress axioms

CAVEAT took 87 seconds to generate and verify the 221 verification conditions necessary to prove the safety property.

7 Conclusions and future work

The invariant validation process often has an interactive part as well as an automatic part [6, 30]. This interactive aspect (even if it is often easy) makes the proof process longer and tedious. This work is one step further to make the proof by invariants more applicable, either as a method by itself, or as an element of an automatic verification process.

The verification conditions obtained in the context of verification of parameterized algorithm are often quantified over the set of processes. We have presented here a quantifier elimination procedure based on an enhanced Herbrand Theorem, an adaptation of the classical Herbrand Theorem to many-sorted logic with equality. This quantifier elimination procedure is suitable for a large class of verification conditions including formulas coming from verification of parameterized systems. It has been successfully applied to the invariant validation for several algorithms included the bakery algorithm (with or without bounded tickets), a railroad crossing system, Burns, Dijkstra, Ricart & Agrawala, Szymanski... As the quantifier-free validity problem is usually decidable, this quantifier elimination procedure is a key to automatic validation of invariants.

With bigger algorithms, instantiation itself may become a problem. Finding simple and effective heuristics to selectively instantiate formulas is also in our concern. A rigorous hypothesis selection and elimination method has already been found in the pure propositional case [19], and the results are promising. We plan to adapt it to the present framework.

References

1. P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In *Computer Aided Verification Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 134–145. Springer-Verlag, July 1999.
2. K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, May 1986.
3. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, July 2001.
4. K. Baukus, Y. Lakhnech, and K. Stahl. Verification of Parameterized Protocols. *Journal of Universal Computer Science*, 7(2):141–158, Feb. 2001.
5. P. Bernays and M. Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Math. Annalen*, 99:342–372, 1928.
6. N. S. Bjørner, Z. Manna, H. B. Sipma, and T. E. Uribe. Deductive verification of real-time systems using STeP. *TCS: Theoretical Computer Science*, 253, 2001.
7. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer-Verlag, July 2000.

8. K. M. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley, Reading, Massachusetts, 1988.
9. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
10. B. Dreben and W. D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas*. Addison-Wesley, Reading, Massachusetts, 1979.
11. E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *Computer Aided Verification*, volume 1102, pages 87–98. Springer-Verlag, July 1996.
12. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.
13. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., Orlando, Florida, 1972.
14. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, Berlin, 1990.
15. P. Fontaine and E. P. Gribomont. Using BDDs with combinations of theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2514 of *Lecture Notes in Computer Science*. Springer, 2002.
16. J. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E–unification. *Journal of the ACM*, 39(2):377–429, Apr. 1992.
17. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992.
18. S. Graf and H. Saïdi. Verifying invariants using theorem proving. In *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 196–207. Springer Verlag, 1996.
19. E. P. Gribomont. Simplification of boolean verification conditions. *Theoretical Computer Science*, 239(1):165–185, May 2000.
20. J. Y. Halpern. Presburger arithmetic with unary predicates is Π_1^1 complete. *The Journal of Symbolic Logic*, 56(2):637–642, June 1991.
21. C. Heitmeyer and N. A. Lynch. The generalized railroad crossing — a case study in formal verification of real-time systems. In *Proceedings 15th IEEE Real-Time Systems Symposium,* San Juan, Puerto Rico, pages 120–131, Dec. 1994.
22. H. E. Jensen and N. A. Lynch. A proof of burns n-process mutual exclusion algorithm using abstraction. In *Tools and Algorithms for Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, Mar. 1998.
23. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435. Springer-Verlag, 1997.
24. R. P. Kurshan and K. McMillan. A structural induction theorem for processes. In *Principles of Distributed Computing*, pages 239–248. ACM Press, Aug. 1989.
25. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, CS, 1996.
26. G. C. Necula. *Compiling with Proofs*. PhD thesis, Carnegie Mellon University, Oct. 1998. Available as Technical Report CMU-CS-98-154.
27. G. Nelson and D. C. Oppen. Simplifications by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, Oct. 1979.
28. A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 82–97, 2001.

29. F. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.
30. N. Shankar. Verification of Real-Time Systems Using PVS. In *Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 280–291. Springer-Verlag, June 1993.
31. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80. Springer-Verlag, June 1989.