Verification of Infinite-State Systems by Combining Abstraction and Reachability Analysis*

Parosh Aziz Abdulla[1] Aurore Annichini[2] Saddek Bensalem[2]
Ahmed Bouajjani[2] Peter Habermehl[3] Yassine Lakhnech[4]

[1] Dept. of Computer Systems, P.O. Box 325, 75105 Uppsala, Sweden.
[2] Verimag, Centre Equation, 2 av. de Vignate, 38610 Gières, France.
[3] Liafa - Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France.
[4] Institut für Informatik und Praktishe Mathematik,
Christian-Albrechts-Universität zu Kiel, Preußerstr. 1-9, 24105 Kiel, Germany.

Abstract. We address the problem of verifying systems operating on different types of variables ranging over infinite domains. We consider in particular systems modeled by means of extended automata communicating through unbounded fifo channels. We develop a general methodology for analyzing such systems based on combining automatic generation of abstract models (not necessarily finite-state) with symbolic reachability analysis. Reachability analysis procedures allow to verify automatically properties at the abstract level as well as to generate auxiliary invariants and accurate abstraction functions that can be used at the concrete level. We propose a realization of this approach in a framework which extends PVS with automatic invariant checking strategies, automatic procedures for generating abstract models, as well as automata-based decision procedures and reachability analysis procedures for fifo channels systems.

1 Introduction

Communication protocols can be naturally modeled by automata communicating through fifo queues. However, in the modeling of such systems, we often need, besides queues, additional variables and data structures such as counters (for instance to memorize the number of messages sent) and timers (to check timeouts and introduce synchrony between processes). Hence, to reason about protocols, we need in general to analyze *extended automata* operating on variables which may range over several different domains. Moreover, the relevant domains may in general be infinite (e.g., in the case of counters, unbounded fifo channels, etc). We are here interested in *automatic* analysis of such *heterogeneous* infinite-state models, especially of extended automata with fifo channels. We develop a general analysis methodology for these models based on combining abstraction and symbolic reachability analysis. We show a realization of this methodology

* This work was partially funded by the National Science Foundation under grant CCR-9509931 to SRI International and has been done while S. Bensalem and P. Habermehl were visiting SRI International.

in a framework which extends the tool InVeSt [6] by automata-based decision procedures and reachability analysis techniques in order to deal with unbounded fifo channels.

Several approaches can be adopted to analyze infinite-state systems. One of them is *abstraction*. It consists in finding an abstraction function allowing to construct a faithful abstract model which can be automatically analyzed [13, 12, 23]. The problem is then how to find a suitable abstraction function and how to derive automatically the abstract model from the concrete one. Several frameworks that provide assistance for performing these tasks [18, 15, 19, 5] have been proposed. However, finding abstraction functions remains in general a non trivial problem which requires a deep understanding of the behavior of the system. In some extreme cases, knowledge of the set of reachable configurations of the system is needed.

Another approach is *symbolic reachability analysis*. It consists in computing a finite representation of the set of all reachable configurations of the system. If such a computation can be done, this approach allows to solve verification problems that are reducible to reachability problems (e.g., verification of safety properties). Moreover, the generation of the set of reachable configurations allows to get for free, and fully automatically, finite abstractions of the system. These abstractions are usually finite transition graphs (called symbolic graphs) where nodes represent sets of configurations. This approach has been applied for several kind of extended automata: timed automata and hybrid automata [4], counter automata [14, 8], pushdown automata [9, 17], fifo-channel automata [3, 7, 10, 2], etc. However, the existing symbolic techniques concern in general *homogeneous* models, i.e., models with one kind of variables ranging over unbounded data domains. So, in order to apply these techniques to communication protocols, we need in general an abstraction step allowing to get homogeneous models from heterogeneous ones.

From the descriptions above of the two approaches, it appears that they are complementary. In this work, we propose to combine these two approaches:

1. Using abstraction and automatic techniques for generating abstract models in order to obtain homogeneous models from heterogeneous ones. Notice that we need here methods allowing to generate automatically abstract extended automata that are not necessarily finite-state.
2. Using symbolic reachability analysis techniques in order to verify automatically properties at the abstract level, as well as to generate auxiliary invariants and abstraction functions that can be used at the concrete level.

Let $\mathcal{A}$ be an extended automaton with variables of two different types T_1 and T_2. To verify a property on $\mathcal{A}$, the ideal situation is that we are able to provide an abstraction function ρ on the whole state space of $\mathcal{A}$, to construct a corresponding finite-state abstract model $\mathcal{A}_\rho$ which simulates $\mathcal{A}$, i.e., that contains for each behavior of $\mathcal{A}$ a corresponding behavior, and to check that the property holds on $\mathcal{A}_\rho$. As we mentioned above, it is in general hard to find such a ρ. However, it is often the case that we have a way to define an abstraction function on the variables of type T_1 (for instance by taking systematically a partition of their space of values according to the predicates appearing in the model), whereas the set of all reachable values of variables of type T2 must still be analyzed precisely.

In such a case, we can start in a first step by applying an abstraction ρ_1 on the variables of type T_1 and obtain a model $\mathcal{A}_{\rho_1}$ where all variables are of type T_2. Then, in a second step, we apply to $\mathcal{A}_{\rho_1}$ a symbolic reachability analysis procedure which is specific to extended automata of type T_2. This procedure computes a representation of $Reach(\mathcal{A}_{\rho_1})$ which is the set of all reachable configurations in $\mathcal{A}_{\rho_1}$. Then, the result of this computation can be used in different ways:

- Generation of invariants and verification of invariance properties: To show that every reachable configuration of $\mathcal{A}$ satisfies a property φ, we can show that $Reach(\mathcal{A}_{\rho_1}) \subseteq \rho_1(\varphi)$ at the abstract level. Moreover, $\rho_1^{-1}(Reach(\mathcal{A}_{\rho_1}))$ is an invariant of the concrete model $\mathcal{A}$ which can help in establishing invariance properties at the concrete level.
- Construction of finite abstractions of $\mathcal{A}_{\rho_1}$: we can consider the partition of $Reach(\mathcal{A}_{\rho_1})$ according to control states, or any finer partition π, and construct the corresponding symbolic graph $(\mathcal{A}_{\rho_1}/\pi)$. This graph can be used to check properties in the universal fragments of temporal logics (in particular, linear-time properties).
- Generation of abstraction functions: Verifying a safety property on $\mathcal{A}$ can be reduced to checking a reachability property on a system $\mathcal{A} \times \mathcal{O}$ where $\mathcal{O}$ is an *observer* (finite automaton expressing the property). Then, each time we consider a new observer $\mathcal{O}$, we need to define an abstraction function on $\mathcal{A} \times \mathcal{O}$. The knowledge of the possible contents of the variables of $\mathcal{A}$ (considered separately) can help in constructing such an abstraction function.

Clearly, in order to apply this methodology, we should have for each type of variables we consider a symbolic representation allowing to represent and manipulate infinite sets of configurations: These structures must allow to perform some basic operations such as boolean operations and the computation of successors and predecessors. We also need decision procedures on these representations for solving emptiness, membership, and entailment problems. These procedures are needed during the symbolic reachability analysis, as well as during the construction of the abstract models for given abstraction functions. Furthermore, since we must reason at the concrete level on heterogeneous models and combine the results of analysis of several kinds of variables, we need a general and uniform framework where all the representation structures we consider can be embedded. In this paper, we consider a framework based on InVeSt-PVS, and we show how this framework is extended in order to support the methodology we describe above for extended automata with fifo queues.

The original framework we consider offers the logical language of PVS [26] which is based on higher order logic, where extended automata can be specified. Also, various decision procedures are available in this framework, in particular for linear arithmetics. The tool InVeSt [6] provides strategies for checking invariants, as well as an automatic procedure for constructing in a compositional manner abstract models for given abstraction functions [5]. During the construction of abstract models, the existing implementation of this procedure in InVeSt invokes PVS (and its decision procedures) in order to decide the existence of the transitions between abstract states. This procedure is reasonably efficient as long as the considered data structures in the models can be described in theories for which PVS provides decision procedures. For instance, this procedure can be applied efficiently in the case of integer counters with linear operations. However, when we consider sequence variables like fifo channels, the use of the PVS-based implementation of this procedure becomes ad-hoc and cumbersome. This is due to the lack in PVS of decision procedures on regular languages. Indeed, sets of contents of sequence variables can be naturally represented by means of finite-state automata or regular expressions, and many of the proof goals concerning these variables could be solved as emptiness or entailment problems of regular languages.

A contribution of this work is to propose an embedding of regular languages in InVeSt-PVS. This embedding consists of a theory of regular expressions allowing to express in the PVS language constraints like $x \in L$ where x is of type sequence and L is a regular language, a connection to the tool AMORE [24] (which provides a library of procedures on finite-state automata), and an automatic procedure for computing abstract models using automata-based decision procedures.

Another contribution of this work is an extension of InVeSt by an automatic reachability analysis procedure for (lossy) fifo-channel systems. This extension is made through a connection to the tool LCS [1] which allows to compute the set of reachable configurations of a lossy fifo-channel system by means of a regular expressions-based symbolic representation, following the procedure introduced in [2]. The tool LCS allows also to generate automatically finite abstract models as symbolic graphs.

We illustrate the use of our framework on the examples of the Alternating Bit Protocol and the Bounded Retransmission Protocol.

2 Extended Fifo-Channel Automata

We consider in this paper *untimed* models of protocols that are parallel compositions of extended automata communicating through unbounded lossy queues.

An extended fifo-channel automaton $\mathcal{A}$ consists of a set of control locations Q, a vector of typed variables $\vec{x}$, an initial control location $q_{init} \in Q$, a set V_{init} of initial vectors of values, and a set of transitions δ between control locations. Each transition is labelled by a *guarded command*. The guard is a predicate on the variables and defines an enabling condition of the transition. The command is a transformation (assignment) of the variables.

Among the variables, we distinguish fifo channels. We suppose that messages in these channels are in a finite alphabet Γ, and we consider the following operations: the emptyness test $empty(c)$ (true if the channel c is empty), $c!a$ (*send* a to channel c), and $c?a$ (*receive* a from c provided a is the first symbol in c), for any $a \in \Gamma$. We do not fix the types of the other variables. They may range over finite or infinite domains (boolean, integers, etc).

A configuration of the system is a tuple $\langle q, \vec{v} \rangle$ where $q \in Q$ is a control location, $\vec{v}$ is a valuation of the variables. Notice that the valuation of each channel is a word on the alphabet Γ. Let $\Sigma_{\mathcal{A}}$ be the set of configurations of $\mathcal{A}$. The set of the *initial configurations* of $\mathcal{A}$ is $Init_{\mathcal{A}} = \{q_{init}\} \times V_{init}$.

The semantics of the model is defined by means of a transition relation $R_{\mathcal{A}} \subseteq \Sigma_{\mathcal{A}} \times \Sigma_{\mathcal{A}}$ between configurations. We assume that the channels are lossy in the sense that they can lose messages at any time. Hence, in the definition of $R_{\mathcal{A}}$, the execution of a transition in δ can be preceded and followed by losses of messages (contents of channels may decrease according to the subsequence ordering). The formal definition of $R_{\mathcal{A}}$ is standard and is omitted here. Then, we associate with $\mathcal{A}$ the transition system $S_{\mathcal{A}} = (\Sigma_{\mathcal{A}}, Init_{\mathcal{A}}, R_{\mathcal{A}})$.

We consider the two usual functions $post_{\mathcal{A}}, pre_{\mathcal{A}} : 2^{\Sigma_{\mathcal{A}}} \to 2^{\Sigma_{\mathcal{A}}}$ such that, for any set of configurations $X \subseteq \Sigma_{\mathcal{A}}$, $post_{\mathcal{A}}(X)$ (resp. $pre_{\mathcal{A}}(X)$) is the set of immediate successors (resp. predecessors) of X in $S_{\mathcal{A}}$. We denote by $\widetilde{post}$ and $\widetilde{pre}$ the dual functions of $post$ and pre, i.e., $\widetilde{\phi} = \neg\phi\neg$, for $\phi \in \{post, pre\}$. We denote by $post^*$ and pre^* the reflexive-transitive closures of $post$ and pre. The set of all *reachable configurations* of $\mathcal{A}$ is defined by $Reach(\mathcal{A}) = post^*(Init_{\mathcal{A}})$.

3 Abstractions and Invariants

Invariants Let $S = (\Sigma, Init, R)$ be a transition system. We say that $\varphi \subseteq \Sigma$ is an *invariant* of S, if $Reach(S) \subseteq \varphi$. Checking that φ is an invariant of S consists in finding an *auxiliary invariant* ψ such that $post_R(\psi) \subseteq \psi$, $Init \subseteq \psi$, and $\psi \subseteq \varphi$.

Of course, one possible ψ is the set $Reach(S)$ itself. However, it is not always possible to have an effective way to construct a representation of $Reach(S)$ in a theory where $Reach(S) \subseteq \varphi$ can be checked effectively. Alternatively, one can use abstractions to prove invariance properties on abstract models for which the set of reachable configurations can be computed in such a theory. This set can also be used to define an auxiliary invariant at the concrete level.

Abstractions Consider two transition systems $(\Sigma_i, Init_i, R_i)$ with $i = 1, 2$. A *Galois connection* [5] between Σ_1 and Σ_2 is a pair (α, γ) of functions $\alpha : 2^{\Sigma_1} \to 2^{\Sigma_2}$ and $\gamma : 2^{\Sigma_2} \to 2^{\Sigma_1}$ such that $\alpha(X_1) \subseteq X_2$ iff $X_1 \subseteq \gamma(X_2)$, for every $X_i \subseteq \Sigma_i$. We call α the *abstraction function* and γ its *concretization*.

Given a Galois connection (α, γ), we say that S_2 is an *abstraction* of S_1, denoted by $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$, if $\alpha(Init_1) \subseteq Init_2$ and $\alpha \circ post_{R_1} \circ \gamma \subseteq post_{R_2}$. We say also in this case that S_2 (α, γ)-simulates S_1. We write $S_1 \sqsubseteq S_2$ (S_2 simulates S_1) if there exists a Galois connection (α, γ) such that $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$.

An efficient way to describe Galois connections consists in giving a total relation $\rho \subseteq \Sigma_1 \times \Sigma_2$. Indeed, it is shown in [23] that such a relation induces the Galois connection $(post_\rho, \widetilde{pre}_\rho)$. It is easy to check that in case ρ is a function, the concretization $\widetilde{pre}_\rho$ coincides with pre_ρ, which we will tacitly denote by ρ^{-1}. In the sequel, we will write $S \sqsubseteq_\rho S_A$ instead of $S \sqsubseteq_{(post_\rho, \widetilde{pre}_\rho)} S_A$. Moreover, we will also refer to $\rho \subseteq \Sigma_1 \times \Sigma_2$ as the *abstraction relation (function)* as the distinction between ρ and $post_\rho$ is easily made from the context.

[5] The use of Galois connections and abstract interpretation as a general and unifying framework for abstraction techniques has been first proposed in [13].

Checking invariance properties under abstraction It can be shown that if $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$ then $Reach(S_1) \subseteq \gamma(Reach(S_2))$. Hence, if $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$ and $Reach(S_2) \subseteq \varphi_2$, then $Reach(S_1) \subseteq \gamma(\varphi_2)$. Therefore, in order to check that $Reach(S_1) \subseteq \varphi_1$, for some $\varphi_1 \subseteq \Sigma_1$, it suffices to find a Galois connection (α, γ) and a system S_2 such that $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$, and to check $post_{R_1}(\gamma(Reach(S_2)) \cap \varphi_1) \subseteq \varphi_1$. Notice, that the last condition holds immediately in case $\gamma(Reach(S_2)) \subseteq \varphi_1$, since $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$ implies $post_{R_1}(\gamma(Reach(S_2))) \subseteq \gamma(Reach(S_2))$. This is the standard preservation result concerning invariance properties [12, 23]. Notice also, that in case φ_1 is an invariant of S_1, there must exist S_2 and (α, γ) which fulfill the conditions above.

Since function composition is monotone and $\subseteq$ is transitive, $S_1 \sqsubseteq_{(\alpha,\gamma)} S_2$ and $S_2 \sqsubseteq_{(\alpha',\gamma')} S_3$ implies $S_1 \sqsubseteq_{(\alpha' \circ \alpha, \gamma \circ \gamma')} S_3$. Then, we can consider a hierarchy of abstractions, that is, a sequence $S_1 \sqsubseteq_{(\alpha_1,\gamma_1)} S_2 \cdots \sqsubseteq_{(\alpha_n,\gamma_n)} S_{n+1}$ with $n \geq 1$, in order to check properties at different levels of abstraction, and derive auxiliary invariants (for every i, $\gamma_1 \circ \cdots \circ \gamma_i(Reach(S_i))$ is an invariant of S_1).

4 Computing Abstract Models

Given a system of extended automata $\mathcal{A} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and an abstraction relation $\rho \subseteq \Sigma \times \Sigma_A$, we want to construct an abstract system $\mathcal{A}_\rho = \mathcal{A}_1^\rho \parallel \cdots \parallel \mathcal{A}_n^\rho$, such that $S_\mathcal{A} \sqsubseteq_\rho S_{\mathcal{A}_\rho}$. For that, we adopt the method presented in [5], which consists in considering separately each concrete transition τ_c, and construct its corresponding abstract transitions τ_a. This is achieved by starting from the universal relation between abstract states and eliminating transitions that do not correspond to concrete ones. Given two abstract states a_1 and a_2 and a concrete transition τ_c of $\mathcal{A}$, if the condition

$$\rho^{-1}(a_1) \Rightarrow \widetilde{pre}_{\tau_c}(\neg \rho^{-1}(a_2)) \tag{1}$$

holds, then the transition τ_a between a_1 and a_2 is removed. In the existing implementation of this method in the InVeSt tool, condition (1) is checked using PVS. In order to enhance the efficiency of the method, it is safe to partition the abstract variables and to compute an abstraction transition for each partition separately. The global abstract transition is then obtained by conjunctively composing these abstract transitions. An interesting property of this technique is that it allows to deal with variables from different types separately, and use for each of these types specific methods and decision procedures to check Condition (1). Moreover, it allows to deal with abstract relations that behave as the identity on variables that range over infinite domains. This is necessary if we want to abstract for instance an extended automaton with counters and fifo channels, without abstracting the channels (we abstract counters and get an unbounded fifo-channels system).

Now, to compute abstract transitions, we need to compute abstractions of concrete guards, as well as concretizations of abstract states. In case the abstraction relation is given by a predicate, these operations involve quantifier elimination which can be computationally costly, when possible. Therefore, we only consider abstraction functions which are given by an expression of the form:$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k_i} (\varphi_{i,j} \to a_i = exp_{i,j})$ where $a_1, \cdots, a_n$ are the abstract variables and the $\varphi_{i,j}$'s partition Σ, for every $i = 1, \cdots, n$, and the $\varphi_{i,j}$'s and $exp_{i,j}$'s only involve concrete variables. Moreover, we require that for every literal l occurring in a guard of some transition of the concrete system there exist i and j such that $\varphi_{i,j}$ is l. This ensures that we can compute an over-approximation of the abstraction of a guard simply by substituting every literal by its corresponding function $a_i = exp_{i,j}$.

Let us now consider the concretization function. If the abstraction relation ρ is given in the form above, then it is a total function and $\widetilde{pre}_\rho(\varphi_A) = pre_\rho(\varphi_A) = \rho^{-1}(\varphi_A)$ is easily computed by $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k_i} (\varphi_{i,j} \rightarrow \varphi_A[exp_{i,j}/a_i])$.

Now, given a concrete model, we can compute an abstraction function ρ which satisfies the requirements above by introducing an abstract boolean variable a_l for each literal l occurring in some guard and defining ρ by the formula $\bigwedge_l a_l = l$. In order to avoid an explosion in the number of abstract variables, literals that refer to the same set of concrete variables are checked whether they build a partitioning of Σ. In case n literals $l_1, \cdots, l_n$ build such a partitioning, a single abstract variable ranging over $\{1, \cdots, n\}$ is introduced instead of n boolean variables. Moreover, it is also possible to consider the predicates occurring in the assignments to obtain a finer abstraction function.

Abstracting queues

The accuracy of the abstract model obtained by applying the method presented above strongly depends on the proof strategy used to check condition (1). In our experience, the use of the decision procedures and proof strategies of PVS leads to reasonable results unless recursive functions and recursive data types are used. Now, since queues range over lists of values, it seems to be natural to encode them using lists and define abstractions on them using recursive functions. This, however, may lead to unnecessarily cumbersome definitions and require ad-hoc proof strategies as is the case for the Alternating Bit Protocol (ABP) and the clever abstraction used by Müller and Nipkow [25].

This abstraction is based on the observation that the content of the channels is always of the form $m_1^* m_2^*$. Hence, if a finite alphabet of messages is considered, one can obtain a finite state abstraction of the ABP by merging adjacent identical messages.

We have specified the ABP in the specification language of PVS. We specified channels as variables of type list over *Mes*, where *Mes* is a finite set of messages. Sending and receiving messages are then specified using list operations as *car*, *cdr*, and *cons*. Müller and Nipkow's abstraction can then be specified using a recursively defined function. Using InVeSt and the proof strategies of PVS, we have computed a finite abstraction of ABP for $Mes = \{0, 1\}$. The difficulty in this exercise has been to find a suitable proof strategy that handles the involved lists and the recursively defined functions. If we had used regular expressions, we could have specified the abstraction function very easily and we could have constructed the abstract system fully automatically without providing any particular strategy using decision procedures on regular languages.

5 Embedding Regular Languages in InVeSt-PVS

To be able to efficiently handle in InVeSt systems with fifo-channels we introduce PVS-theories for queues and regular languages, and we embed automata-based decision procedures. These extensions allow us to naturally represent sets of contents (sequences of messages) of fifo channels by means of regular expressions, which simplifies the definition of abstractions functions on queues as well as the construction of the corresponding abstract models.

5.1 Extending the specification language of PVS

A theory for queues is introduced on the theory of finite sequences which is predefined in PVS. This new theory includes the definition of a polymorphic type $queue[Mes]$ and the definitions of the polymorphic functions add, $front$ and $remove$. Using these functions, sending a message (resp. receiving a message) can be specified by the guarded commands $true \rightarrow c := add(m, c)$ (resp. $front(c) = m \rightarrow c := remove(c)$).

Then, we introduce a theory to deal with extended regular expressions with the standard operations (Kleene star, concatenation, union), as well as positive Kleene star $(\cdot^{+})$, intersection, complementation and right-quotient $(\cdot^{-1})$. This theory allows to express language constraints like $c \in L$ where c is a queue variable and L a language given by an extended regular expression on the set of message symbols Mes. Using this theory, we can specify abstraction functions on queues by a formula of the form: $\bigwedge_{i=1}^{n} c \in L_i \Rightarrow c_A = a_i$, where $Chcont^a = \{a_1, \ldots, a_n\}$ is a finite set of abstract values, c_A is the abstract variable associated with the queue c, and $L_1, \cdots, L_n$ are regular expressions which form a partition of Mes^*. For example, the Müller-Nipkow abstraction function on the channels of the ABP (see section 4) can be defined straightforwardly using this theory $(c \in 0^+ \cdot 1^+ \Rightarrow c_A = 01 \;\wedge\; c \in 1^+ \cdot 0^+ \Rightarrow c_A = 10 \;\wedge\; \cdots)$. Notice that our PVS theory on regular expressions is also useful for the representation of sets of reachable states calculated symbolically (see Section 6).

5.2 Calculating the abstract operations on a fifo-channel

Let Mes be a set of messages, let c be a queue variable, and let ρ be its abstraction function defined by a formula as above. Following the method described in Section 4, given an operation op on c (i.e., $op \in \{c!m, c?m \mid m \in Mes\}$), we compute the abstract operation op_A by checking conditions of the form (1), which is equivalent to check the complementation-free condition $\rho^{-1}(c_A = a_i) \cap pre_{op}(\rho^{-1}(c_A = a_j)) = \emptyset$. By definition of ρ, $\rho^{-1}(c_A = a_i)$ corresponds to L_i (represented in our PVS theory by the predicate $c \in L_i$). Hence, the condition above is equivalent to $L_i \cap pre_{op}(L_j) = \emptyset$. It easy to see that pre_{op} can be defined in terms of basic operations on regular languages: $pre_{c!m}(L) = L \cdot m^{-1}$ and $pre_{c?m}(L) = m \cdot L$. Hence, checking the condition above consists in checking emptiness problems on regular languages, which can be done automatically by invoking decision procedures for extended regular expressions. For that, we have connected InVeSt to the tool Amore [24] which can handle regular expressions and decide problems like emptiness, inclusion, etc.

We repeated the ABP example using our PVS theories on queues and regular expressions, and the InVeSt-AMORE connection. Not only writing the abstraction function is significantly simpler within this framework, but also the time for computing the abstract models reduced from ~ 45 minutes to 11 seconds.

6 Reachability Analysis: The tool Lcs (Lossy Channel Systems)

6.1 Computing reachability sets

The set $Reach(\mathcal{A})$ of any lossy fifo-channel automaton is recognizable but not effectively constructible (there is no algorithm allowing to compute a representation of this set for any $\mathcal{A}$) [11]. Hence, we adopt a semi-algorithmic approach based on computing successively representations of an increasing sequence of (lower) approximations of $Reach(\mathcal{A})$, by adding at each step the immediate successors (*post*-images) of the configurations computed so far, and using *acceleration* techniques [22, 13] in order to enhance the chance of convergence. When our procedure terminates, it delivers a structure representing precisely the set $Reach(\mathcal{A})$. The acceleration principle we adopt is based on computing in one step the effect of executing a control loop an arbitrary number of times (control loops are considered as *meta-transitions* [8, 7]).

To realize this approach, we need symbolic representation structures of sets of configurations which allow *finite* representations of the infinite sets we are interested in, which are effectively closed under union and *post*, which have a decidable entailment problem, and moreover, which allow the computation and the representation of images by meta-transitions (the effects of control loops). Another important feature of such a representation structure is to be *normalizable* formal, i.e., for every representable set, there is a unique normal (or canonical) representation which can be derived from any alternative representation. Indeed, all operations (e.g., entailment testing) are often easier to perform on normal forms. Furthermore, normality (canonicity) often corresponds to a notion of minimality, which is crucial for practical reachability analysis procedures.

In [2], we have introduced a symbolic representation structure based on a class of regular expressions called SRE's, for use in the computation of the sets of reachable configurations of lossy fifo-channels automata. An SRE is either the empty set $\emptyset$ or a finite union of products, each of these products is either an empty string ϵ or a finite concatenation $e_1 \cdots e_n$ of atomic expressions which can be either of the form $(a + \epsilon)$, or $(a_1 + \cdots + a_n)^*$. We showed in [2] that SRE's satisfy all the needed features mentioned above: they characterize exactly the class of reachability sets of lossy channels automata, and there are simple and efficient procedures (polynomial) for normalization, for entailment testing, for computing *post*-images, and for computing the effect of *any* control loop.

Based on the results of [2], we have implemented in a tool called LCS [1] a procedure for computing reachability sets using the following principle: Given a parallel composition of a set of fifo-channel automata, the procedure starts from the initial configurations and constructs the set of all reachable configurations by applying a depth-first-search strategy through a symbolic transition graph. The nodes of this graph are *symbolic states*, i.e., representations of sets of configurations as pairs of the form $\langle q, E \rangle$, where q is a control location and E is an SRE-based representation of the contents of the channels: If the system has n channels, E is a finite set $\{(e_1^1, \ldots, e_n^1), \ldots, (e_1^m, \ldots, e_n^m)\}$ of n-dim vectors of SRE's representing the set $[\![E]\!] = \bigcup_{i=1}^{m} [\![e_1^i]\!] \times \cdots \times [\![e_n^i]\!]$, where $[\![e]\!]$ denotes the language described by the expression e. At each step, the procedure computes the immediate successors (post-images) of the current symbolic state by all possible transitions of the automaton, and considers them according to the depth-first-search ordering. When a control loop is detected, an acceleration is performed by computing the effect of iterating the considered control loop on the current symbolic state. The set of encountered configurations is memorized progressively. After computing the post-image of a symbolic state, the procedure checks whether the obtained symbolic state is covered by the set of configurations computed so far. If this is the case, the successors of this symbolic state are not generated. Notice that this procedure can also be used for *on-the-fly* verification of safety or invariance properties.

6.2 Constructing Finite Abstract Models

Computing the set of reachable configurations can be used to generate finite abstract models. Let $\mathcal{A}$ be a fifo-channel automaton. Let Φ be a finite set of symbolic states of $\mathcal{A}$ (see definition in the previous paragraph). Then, the symbolic graph associated with Φ is the finite-state transition system $\mathcal{G}_\Phi = (\Phi, Init_\Phi, R_\Phi)$ such that $Init_\Phi$ is the set of all symbolic states containing the initial configuration $init_\mathcal{A}$, and $\forall \phi_1, \phi_2 \in \Phi.\ \phi_1 R_\Phi \phi_2$ iff $\exists \sigma_1 \in \phi_1, \sigma_2 \in \phi_2.\ \sigma_1 R_\mathcal{A} \sigma_2$.

The *canonical symbolic graph* of $\mathcal{A}$ corresponds to the partition of $Reach(\mathcal{A})$ according to the control states, i.e., $\Phi_\mathcal{A} = \{\langle q_1, E_1 \rangle, \ldots, \langle q_m, E_m \rangle\}$ where $Q = \{q_1, \ldots, q_m\}$ and $Reach(\mathcal{A}) = \bigcup_{i=1}^{m} \{q_i\} \times [\![E_i]\!]$. It is easy to see that for every set of symbolic states which covers $Reach(\mathcal{A})$, i.e., $Reach(\mathcal{A}) \subseteq \bigcup_{\phi \in \Phi} \phi$, we have $S_\mathcal{A} \sqsubseteq \mathcal{G}_\Phi$ ($\mathcal{G}_\Phi$ simulates $S_\mathcal{A}$). This fact holds in particular for the canonical symbolic graph $\mathcal{G}_{\Phi_\mathcal{A}}$, as well as for all the symbolic graphs obtained from refinements of the partition $\Phi_\mathcal{A}$.

The tool LCS allows the automatic construction of the canonical symbolic graph of a given fifo-channel system. The construction of this graph is done during the construction of the reachability set.

Example: The Alternating Bit Protocol can be modeled by two automata, a sender and a receiver, communicating through two lossy channels K and L. We applied to the ABP the procedure of the LCS tool which has terminated and generated the set of reachable configurations given in the table 1, as well as the canonical symbolic graph. The execution time is 0.07 seconds (UltraSparc). This symbolic graph is then reduced after hiding all internal actions to a cyclic graph with two transitions, a SND followed by a RCV, which shows that the protocol behaves as a one-place buffer.

Sender	Receiver	Chan. K	Chan. L	Sender	Receiver	Chan. K	Chan. L
0	0	1^*	1^*	2	2	0^*	0^*
1	0	1^*0^*	1^*	3	2	0^*1^*	0^*
1	1	0^*	1^*	3	3	1^*	0^*
1	2	0^*	1^*0^*	3	0	1^*	0^*1^*

Table 1. Reachability set of the ABP

7 Combining Abstraction and Reachability Analysis

7.1 From the Concrete to the Abstract $\cdots$

The first possible combination of abstraction and reachability analysis is to apply these two techniques sequentially. Given an heterogeneous model $\mathcal{A}$ of a system, say a parallel composition of extended automata with counters and fifo-channels, the first step consists in applying an abstraction $\mathcal{A}_\rho$ in order to get a (unbounded) fifo-channels system, and then the second step consists in applying the symbolic reachability analysis method in order to, either check directly (on-the-fly) an invariance property on the abstract fifo-channel model $\mathcal{A}_\rho$, or to generate a finite abstraction of this model which can be used for finite-state model-checking.

We illustrate this approach on the example of the Bounded Retransmission Protocol (BRP). Detailed descriptions of the BRP can be found in [21, 20, 16]. The BRP is a data link protocol whose service consists in transmitting large files (sequences of data of arbitrary lengths) from one client to another one. Each datum is transferred in a separate frame. Both clients, the sender and the receiver, obtain an indication whether the whole file has been delivered successfully or not. We model this protocol by means of two automata, a sender and a receiver, communicating through two lossy channels K and L. The BRP can be seen as an extended version of the ABP. However, one of the specific features of the BRP is that the model of the sender uses integer counters: First, it has a counter which gives the index i of the current frame in the considered file. This counter allows to know whether the current frame is the first one, the last one, or some intermediate frame. When the sender does not receive an acknowledgment, it may resend the same message up to a maximal number of retransmissions MAX which is a parameter of the protocol. Hence, the sender uses a counter CR for counting retransmissions.

Starting from this model, we use InVeSt-PVS to generate automatically an abstract model which is an unbounded fifo-channel system. For that, we consider an abstraction function on the integer variables and parameters. This abstraction function is defined *automatically* from the guards appearing in the model (as shown in section 4). Then, the corresponding abstract unbounded fifo-channel model is computed automatically and compositionally. The execution times for computing the abstract sender and receiver are 64.71 and 10.47 sec.

Then, in a second step, we apply the Lcs tool to the obtained abstract fifo-channel model. The Lcs tool constructs automatically the set of reachable configurations and the canonical symbolic graph. The execution time for these operations is 0.56 seconds (UltraSparc). After hiding internal actions and minimization, we obtain a finite transition system (5 states and 10 transitions) which is used to model-check service properties of the BRP such as: between two consecutive requests, the sender and the receiver must deliver indications of success or failure to their clients, the receiver delivers a failure indication only if an abortion (by the sender) has occurred, etc.

7.2 ··· and Back

Strengthening of Invariants Suppose that we want to show that all configurations of $\mathcal{A}$ satisfy a property φ expressed as a predicate on the variables of $\mathcal{A}$. Then, given an abstraction function ρ, we can consider the corresponding abstract model $\mathcal{A}_\rho$ and compute $\mathit{Reach}(\mathcal{A}_\rho)$. Since, $\rho^{-1}(\mathit{Reach}(\mathcal{A}_\rho))$ is an invariant of $\mathcal{A}$ (see Section 3), to solve our verification problem on $\mathcal{A}$ it suffices to show that $\mathit{post}_{\mathcal{A}}(\rho^{-1}(\mathit{Reach}(\mathcal{A}_\rho)) \wedge \varphi) \Rightarrow \varphi$.

Let $\mathit{Reach}(\mathcal{A}_\rho)) = \{\langle q_i, e_1^i, \ldots, e_n^i\rangle \mid i = 1, \ldots, m\}$. Notice that the e_i^j's may be empty sets (when the q_i's are not reachable). Notice also that control locations in $\mathcal{A}_\rho$ correspond to finite abstractions of variables (e.g., counters) in $\mathcal{A}$. Then, the concretization $\rho^{-1}(\mathit{Reach}(\mathcal{A}_\rho))$ is given by the formula $\bigwedge_{i=1}^m(\rho^{-1}(q_i) \Rightarrow \bigwedge_{j=1}^n c_j \in e_j^i)$, which can be written in PVS using our theory on regular expressions. Now, it is worth to notice that in general the formula φ we want to check does not constrain the contents of the channels. The conjunction of the formula above with φ allows to strengthen this formula according to the fact that some control locations in $\mathcal{A}_\rho$ are not reachable and expresses constraints on the variables of $\mathcal{A}$ that are not channels.

Generating and Reusing Abstraction Functions Safety properties can be expressed by means of observers. An observer $\mathcal{O}$ is an extended automaton which runs in parallel with the system and observes its behaviors without interfering with them. Then, invariance properties can be checked on the synchronous parallel composition $\mathcal{A} \times \mathcal{O}$ of the system and its observer. In general, we may consider the same system $\mathcal{A}$ with several observers. Then, it is interesting to compute informations about $\mathcal{A}$ that can be reused each time we consider a composed system $\mathcal{A} \times \mathcal{O}$. In particular, we can use our symbolic reachability analysis of fifo-channel systems in order to derive informations on the contents of the channels of $\mathcal{A}$ (notice that usually, observers are not fifo-channel systems since they represent service properties. However, they may have unbounded local variables like counters). If $\mathcal{A}$ is itself an heterogeneous system, to obtain the information about the contents of the channels of $\mathcal{A}$, we start by applying an abstraction in order to get a fifo-channel system $\mathcal{A}_\rho$ and we compute $\mathit{Reach}(\mathcal{A}_\rho)$. Then, the information we compute allows to define once and for all an abstraction function on the channels which can be used, each time we consider an observer $\mathcal{O}$, in the definition of a finite abstraction of the system $\mathcal{A} \times \mathcal{O}$.

Indeed, given a description of the set of reachable configurations of a fifo-channel systems $\mathcal{A}_\rho$, we can define systematically an abstraction function on its channels: Let $Reach(\mathcal{A}_\rho)) = \{\langle q_i, e_1^i, \ldots, e_n^i\rangle \mid i = 1, \ldots, m\}$. Then, for each $i = 1, \ldots n$, let $\mathcal{C}_i = \{e_i^1, \ldots, e_i^m\}$, and let $\mathcal{R}_i = \bigcup_{j=1}^m e_i^j$ (the set of all possible contents of c_i). We let P_i denote the coarsest partition of $\mathcal{R}_i$ which is compatible with the collection $\mathcal{C}_i$ (i.e., $\forall e \in \mathcal{C}_i.\ \exists p_1, \ldots, p_k \in P_i.\ e = p_1 \cup \ldots \cup p_k$) and we consider a finite set of *abstract contents* $A_i = \{a_p \mid p \in P_i\}$. Then, we define the abstraction function $\rho_i : \Gamma^* \to A_i \cup \{\bot\}$ such that: $\forall w \in \Gamma^*.\ (\bigwedge_{p \in P_i} w \in p \Rightarrow \rho_i(w) = a_p) \wedge w \notin \mathcal{R}_i \Rightarrow \rho_i(w) = \bot$.

Notice that the abstraction functions we generate this way can be written in the PVS language using our theory on regular expressions, and hence, they can be composed with other abstraction functions concerning other variables.

As an illustration, consider again the example of the ABP. Starting from the set of reachable configurations given in Table 1 which was computed by the LCS tool, we generate using the definition above abstraction functions ρ_K and ρ_L for the channels K and L. These two functions are equal and coincide exactly with the Müller-Nipkow abstraction function (see Sections 4 and 5). Based on this abstraction function, we can define an abstraction function of the ABP composed with an observer which checks that the input and output streams coincide.

8 Conclusion

We have developed a methodology for verifying infinite-state systems by combining automatic abstraction techniques and symbolic reachability analysis procedures. We have illustrated the application of this methodology on the case of extended fifo-channels systems. For that, we have extended the tool InVeSt by automata-based decision procedures and reachability analysis techniques for fifo-channels systems.

The method we propose for combining abstractions and reachability analysis can be applied for any type of variables or combination of (interdependent) types corresponding to decidable (mixed) theories, and for which there are symbolic representations and procedures for reachability analysis. Hence a crucial issue is to identify such decidable theories and the corresponding representation structures, and to design efficient symbolic reachability analysis procedures based on these representations. Indeed, improving the power and the efficiency of these procedures allows to simplify the needed abstraction steps.

References

1. P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol. In *TACAS'99*. LNCS 1579, 1999.
2. P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *CAV'98*. LNCS 1427, 1998.
3. P.A. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. *Inform. and Comput.*, 127(2):91–101, 1996.
4. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *TCS*, 138, 1995.

5. S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In *CAV'98*. LNCS 1427, 1998.
6. S. Bensalem, Y. Lakhnech, and S. Owre. InVeSt : A Tool for the Verification of Invariants. In *CAV'98*. LNCS 1427, 1998.
7. B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. In *CAV'96*. LNCS 1102, 1996.
8. B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In *CAV'94*. LNCS 818, 1994.
9. A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.
10. A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. In *ICALP'97*. LNCS 1256, 1997.
11. Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Inform. and Comput.*, 124(1):20–31, 1996.
12. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM TOPLAS*, 16(5), 1994.
13. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Recursive Procedures. In *IFIP Conf. on Formal Description of Programming Concepts*. North-Holland Pub., 1977.
14. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *POPL'78*. ACM, 1978.
15. D. Dams, R. Gerth, and O. Grumberg. Generation of Reduced Models for Checking Fragments of CTL. In *CAV'93*. LNCS 697, 1993.
16. P. D'Argenio, J-P. Katoen, T. Ruys, and G.J. Tretmans. The Bounded Retransmission Protocol must be on Time. In *TACAS'97*. LNCS 1217, 1997.
17. A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*, 1997.
18. S. Graf and C. Loiseaux. A Tool for Symbolic Program Verification and Abstraction. In *CAV'93*. LNCS 697, 1993.
19. S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In *CAV'97*, volume 1254 of *LNCS*, 1997.
20. J-F. Groote and J. Van de Pol. A Bounded Retransmission Protocol for Large Data Packets. In *AMAST'96*. LNCS 1101, 1996.
21. L. Helmink, M.P.A. Sellink, and F. Vaandrager. Proof checking a Data Link Protocol. In *Types for Proofs and Programs*. LNCS 806, 1994.
22. R.M. Karp and R.E. Miller. Parallel Program Schemata: A Mathematical Model for Parallel Computation. In *Switch. and Automata Theory Symp.* IEEE, 1967.
23. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property Preserving Abstractions for the Verification of Concurrent Systems. *FMSD*, 6(1), 1995.
24. Oliver Matz, Axel Miller, Andreas Potthoff, Wolfgang Thomas, and Erich Valkema. Report on the Program AMoRE. Technical Report 9507, Inst. f. Informatik u. Prakt. Math., CAU Kiel, 1995.
25. O. Müller and T. Nipkow. Combining Model Checking and Deduction for I/O-Automata. In *TACAS'95*. LNCS 1019, 1995.
26. 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, Feb. 1995.