An Invariant-Based Approach to the Verification of Asynchronous Parameterized Networks

Igor V. Konnov and Vladimir A. Zakharov

Faculty of Computational Mathematics and Cybernetics, Lomonosov State University, Moscow, RU-119899, Russia
kkonnov@cs.msu.su, zakh@cs.msu.su

Abstract. A uniform verification problem for parameterized systems is to determine whether a temporal property is true for every instance of the system which is composed of an arbitrary number of homogeneous processes. To cope with this problem we combine an induction-based technique for invariant generation and conventional model checking of finite state systems. At the first stage of verification we try to select automatically an appropriate invariant process which is greater (with respect to some preorder relation) than any instance of the parameterized system. At the second stage, as soon as an invariant of the parameterized system is obtained, we verify it by means of conventional model checking tool for temporal logics. To demonstrate the feasibility of quasi-block simulation we implemented this technique and applied it to verification of Resource ReSerVation Protocol (RSVP).

Keywords: program verification, asynchronous networks, invariant generation, induction, simulation, model checking.

1 Introduction

Verification plays an important role in designing reliable computer systems. Two main approaches to program verification are testing and formal verification. Since the behavior of concurrent systems is usually very complicated and tends to be non-reproducible, many bugs are difficult to detect by conventional testing. Formal verification approach provides a more preferred solution. It assumes that one builds a mathematical model for the system to be analyzed, specifies the properties the system should comply with, and then applies some appropriate mathematical techniques to check that the model satisfies the properties. Formal verification is relatively inexpensive in comparison to exhaustive simulation; it receives an ample algorithmic support from various branches of mathematics and manifests its strength in areas where other verification methods are inadequate.

Formal methods fall into the following major categories: model checking, theorem proving, and equivalence checking. Model checking allows verification of computer system by checking that a model $M(P)$ (usually represented as transition system derived from hardware or software design $P$) satisfies a formal specification $\varphi$ (usually represented as temporal logic formula). When $M(P)$ is a finite state model then one could find a rich variety of model checking
procedures (see [9]). In what follows we will assume that each system (process) \( P \) under consideration has only finite state and will not distinguish it from its model (transition system) \( M(P) \).

The development of effective techniques for checking parameterized systems \( \mathcal{F} = \{ P_k \}_{k=1}^{\infty} \) is one of the most challenging problems in verification today. The parameter \( k \) may stand for any variable characteristics of the design \( P_k \) (say, the size of some data structures, stacks, etc.), but much attention is given to the cases when the concurrent systems \( P_k \) are the parallel compositions \( p_1 \parallel p_2 \parallel \ldots \parallel p_k \parallel q \) of similar "user" processes \( p_1, p_2, \ldots, p_k \) and a control process ("environment") \( q \). Then the uniform verification problem for parameterized systems is formulated as follows: given an infinite family \( \mathcal{F} \) of systems \( P_k = p_1 \parallel p_2 \parallel \ldots \parallel p_k \parallel q \) and a temporal formula \( \varphi \), check that each transition system \( M(P_k) \) satisfies \( \varphi \).

Though in [1] it was shown that the problem is undecidable in general, some positive results may be obtained for specific parameterized systems. For the most part three basic techniques, namely, symmetry, abstraction, and induction are employed to extend the applicability of conventional model checking to restricted families of parameterized systems.

The idea of exploiting symmetry for state set reduction was introduced in [6, 15, 16]. Symmetry-based reduction has been successfully applied to a number of case studies (see [4] for survey) and now it is implemented within a framework of many model-checkers [2, 19]. However, in many practical cases this approach run into obstacles, since the problem of finding orbit representatives is as hard as graph isomorphism problem. Some papers [14, 24] have demonstrated a considerable progress in automatic symmetry detection, but this problem still remains the main critical point of the symmetry-based reduction techniques.

Abstraction is likely to be the most important technique for coping with state explosion problem in model checking. A theoretical framework for abstraction technique has been developed in [5, 13, 20]. Abstraction has been widely applied in verification of parameterized systems. Only with the essential help of abstraction does it become possible to apply model checking to verify infinite state systems (see [7, 23]). But, unfortunately, most of the abstraction techniques require user assistance in providing key elements and mappings.

The common idea of the induction technique can be summarized as follows. Define some preorder \( \preceq \) (a simulation or bisimulation) on transition systems and choose some class of temporal formulae \( \text{Form} \) such that

1. the composition operator \( \parallel \) is monotonic w.r.t. \( \preceq \), i.e. \( P_1 \preceq P'_1 \) and \( P_2 \preceq P'_2 \) imply \( P_1 \parallel P_2 \preceq P'_1 \parallel P'_2 \);
2. the preorder \( \preceq \) preserves the satisfiability of formulae \( \varphi \) from \( \text{Form} \), i.e. \( P' \models \varphi \) and \( M \preceq P' \) imply \( M \models \varphi \).

Then, given an infinite family \( \mathcal{F} = \{ P_k \}_{k=1}^{\infty} \), where \( P_k = p_1 \parallel p_2 \parallel \ldots \parallel p_k \parallel q \), find a finite transition system \( \mathcal{I} \) such that

3. \( P_n \preceq \mathcal{I} \) for some \( n, \ n \geq 1 \);
4. \( p_i \parallel \mathcal{I} \preceq \mathcal{I} \).
A transition system \( \mathcal{I} \) which meets the requirements 3 and 4 is called an invariant of the infinite family \( \mathcal{F} \). Requirements 1, 3 and 4 guarantee that \( P_k \preceq \mathcal{I} \) holds for every \( k, k \geq n \). If a property is expressed by a formula \( \varphi \) from Form then, in view of the requirement 2, to verify this property of the parameterized system \( \mathcal{F} \) it is sufficient to model check \( I \) and \( P_k, 1 \leq k < n \), against \( \varphi \). The latter may be done by means of any suitable model-checking techniques for finite state transition systems. This approach to the verification of parameterized networks was introduced in [22, 31] and developed in many papers (see [4] for a survey).

The central problem with induction technique is that of deriving a general method for constructing invariants automatically. In many cases invariants can be obtained by application of the following heuristics: if \( P_{k+1} \preceq P_k \) holds for some \( k \) then \( P_k \) may be used as an invariant \( \mathcal{I} \). This consideration captures formally the common belief that some “typical” instance of a parameterized system inherits the most essential features of the whole family. This idea was applied in [7, 8, 15] for developing fully automated approach for verifying parameterized networks.

The right choice of a preorder \( \preceq \) is of prime importance for the successful application of this heuristic in practice. In [7] and [15] strong simulation and block bisimulation respectively were used as a preorder \( \preceq \), but so far as we know no systematic study of other possible preorders has been made (though bisimulation equivalences were studied in detail). It is clear that the weaker is preorder \( \preceq \), the larger in number are the cases of parameterized systems for which this approach succeeds. In [21] we introduced a block simulation preorder (which is an amalgamation of block bisimulation [15] and visible simulation [18]) and applied this preorder to generate straightforwardly invariants of some parameterized systems. In this paper we continue this line of research. Since asynchronous composition of processes is not monotonic w.r.t. block simulation in general case, we extend this preorder and introduce a quasi-block simulation which is weaker than block simulation. We show that quasi-block simulation preserves the satisfiability of formulae from \( ACTL^*-X \) and that asynchronous composition of processes is monotonic w.r.t. quasi-block simulation. This suggests the use of quasi-block simulation for generating invariants in the induction-based verification techniques. We implemented this technique and applied it to verification of Resource ReSerVation Protocol (RSVP). Hitherto formal verification of RSVP has been studied in [12] with the help of CSP and in [30] in the framework of Coloured Petri Nets. We demonstrate that induction-based verification of RSVP can be performed by employing quasi-block simulation.

The paper is organized as follows. In Section 2 we define the basic notions, including asynchronous composition of labelled transition systems, block and quasi-block simulations on transition systems. In Section 3 we study some essential features of quasi-block simulation. The most important property of quasi-block simulation is its monotonicity w.r.t. parallel composition of labelled transition systems. In Section 4 we consider RSVP as a case study and demonstrate how to verify this protocol using induction technique based on quasi-block simulation for generating a suitable invariant. Section 5 concludes with some directions for future research.
2 Definitions

Definition 1. Labelled Transition System (LTS) is a sextuple $M = \langle S, S_0, A, R, \Sigma, L \rangle$ where

- $S$ is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- $A$ is a set of visible actions (not including the invisible action $\tau$),
- $R \subseteq S \times A \cup \{\tau\} \times S$ is a labelled transition relation,
- $\Sigma$ is a nonempty set of atomic propositions,
- $L : S \rightarrow 2^\Sigma$ is an evaluation function on the set of states.

Any triple $(s, a, t)$ from $R$ is called a transition. We will write $s \xrightarrow{a}_M t$ instead of $(s, a, t) \in R$ and often elide the subscript $M$ when it is assumed. A path $\pi$ of LTS $M$ is a finite or infinite sequence $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_j} s_j \xrightarrow{a_j} \ldots$ of transitions $(s_i \xrightarrow{a_i} s_{i+1})$.

Temporal specifications (or properties) of parameterized systems are expressed in temporal logics. The logics used in the framework of induction-based verification technique are usually the Full Branching Time Logic $CTL^*$ or its sub-logics $ACTL^*$ and $ACTL^*_{\text{X}}$. An important factor in deciding between them is the capability of a preorder $\preceq$ used in verification procedure to preserve the satisfiability of temporal formulae. We do not define the syntax and the semantics of these logics; they may be found in many textbooks, e.g. in [9].

Let $M_1$ and $M_2$ be two LTS’s, $M_i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i \rangle$, $i = 1, 2$, such that $\Sigma^1 \cap \Sigma^2 = \emptyset$. We call a synchronizer any pair $\Gamma = (\Delta, \pi)$, where $\Delta \subseteq A^1$, and $\pi : \Delta \rightarrow A^2$ is an injection which binds some actions of $M_1$ and $M_2$. We write $\Delta$ for the set $\{b \in A^2 \mid \exists a \in \Delta : \pi = b\}$. When introducing a synchronizer we assume that some actions $a$ are executed only synchronously with the co-actions $\pi$. Thus, a pair $(a, \pi)$ forms a channel for communication between $M_1$ and $M_2$. One of this action (say, $a$) may be thought as an action of sending a message, whereas the other (co-action $\pi$) is an action of receiving a message.

Definition 2. The (asynchronous) parallel composition of LTS’s $M_1$ and $M_2$ w.r.t. synchronizer $\Gamma$ is an LTS $M = M_1 \parallel R M_2 = \langle S, S_0, A, R, \Sigma, L \rangle$ such that

- $S = S^1 \times S^2$, $S_0 = S^1_0 \times S^2_0$, $A = A^1 \cup A^2 \setminus (\Delta \cup \Delta^\text{op})$, $\Sigma = \Sigma^1 \cup \Sigma^2$, $L(s, u) = L^1(s) \cup L^2(u)$
- For every pair of states $(s, u), (t, v) \in S$ and an action $a \in A$ a transition $((s, u), a, (t, v))$ is in $R$ if one of the following requirements is met:
  - $a \in A^1 \setminus \Delta$, $u = v$, $(s, a, t) \in R^1$ ($M_1$ executes $a$),
  - $a \in A^2 \setminus \Delta^\text{op}$, $s = t$, $(u, a, v) \in R^2$ ($M_2$ executes $a$),
  - $a = \tau$, and there exists $b \in \Delta$ such that $(s, b, t) \in R^1$ and $(u, b, v) \in R^2$ ($M_1$ and $M_2$ communicate),

Let $\varphi$ be a temporal formula. Denote by $\Sigma_\varphi$ the set of all basic propositions involved in $\varphi$. Given an LTS $M = \langle S, S_0, A, R, \Sigma, L \rangle$, one may separate those
transitions of $M$ that either are visible (i.e. marked with an action $a \neq \tau$) or affect the basic propositions of $\varphi$:

$$\text{Observ}(M, \Sigma_\varphi) = \{(s, a, t) | (s, a, t) \in R \text{ and } (a \neq \tau \lor L(s) \cap \Sigma_\varphi \neq L(t) \cap \Sigma_\varphi)\}.$$ 

On the other hand, one may also distinguish some set of transitions that seemed “significant” for an observer. Any set $E \subseteq R$ of transitions which includes all visible transitions will be called a set of events of $M$. If $\text{Observ}(M, \Sigma_\varphi) \subseteq E$ then the set of events $E$ will be called well-formed w.r.t. $\varphi$.

**Definition 3.** A finite block from a state $s_1$ w.r.t. a set of events $E$ is a finite path $B = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1}$ such that $(s_m, a, s_{m+1}) \in E$ and $(s_1, \tau, s_{i+1}) \notin E$ for all $i : 1 \leq i < m$. An infinite block from a state $s_1$ is an infinite sequence $B = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{\tau} \cdots$ such that $(s_i, \tau, s_{i+1}) \notin E$ for all $i \geq 1$.

We write $\text{MAXF}(E, s)$ and $\text{MAXI}(E, s)$ for the set of all finite and infinite blocks, respectively, from a state $s$ w.r.t. a set of events $E$.

**Definition 4.** Let $M_i = (S_i, S_0^i, A^i, R^i, \Sigma^i, L^i)$, $i = 1, 2$, be two LTSs, let $\Sigma_0$ be a subset of $\Sigma^1 \cap \Sigma^2$, and let $E^1$ and $E^2$ be some sets of events of $M_1$ and $M_2$. Then a binary relation $H \subseteq S^1 \times S^2$ is called a quasi-block simulation (qbsimulation) on $M_1$ and $M_2$ w.r.t. $\Sigma_0$, $E^1$, $E^2$, iff for every pair $(s_1, t_1) \in H$ meets the following requirements:

1. $L^1(s_1) \cap \Sigma_0 = L^2(t_1) \cap \Sigma_0$.
2. For every finite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1} \in \text{MAXF}(E^1, s_1)$ there is a block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{a} t_{n+1} \in \text{MAXF}(E^2, t_1)$ such that $(s_{m+1}, t_{n+1}) \in H$, and $(s_i, t_j) \in H$ holds for every pair $i, j$, $1 \leq i \leq m$, $1 \leq j \leq n$.
3. For every infinite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{\tau} \cdots \in \text{MAXI}(E^1, s_1)$ there is an infinite block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{\tau} \cdots \in \text{MAXI}(E^2, t_1)$, such that $(s_i, t_j) \in H$ holds for every pair $i, j$, $1 \leq i, j \leq 1$.

We write $M_1 \preceq_{\Sigma_0} M_2$ iff there exist two sets of events $E^1$ and $E^2$ of LTSs $M_1$ and $M_2$ and a binary relation $H \subseteq S^1 \times S^2$ such that $H$ is a qbsimulation on $M_1$ and $M_2$ w.r.t. $\Sigma_0$, $E^1$, $E^2$, and for every initial state $s_0 \in S_0^1$ there exists an initial state $t_0 \in S_0^2$ such that $(s_0, t_0) \in H$. If both $E^1$ and $E^2$ are well-formed w.r.t. $\varphi$ then we say that the qbsimulation $M_1 \preceq_{\Sigma_0} M_2$ is also well-formed w.r.t. $\varphi$. A block simulation $(M_1 \preceq_{\Sigma_0} M_2)$ in symbols is a qbsimulation w.r.t. $\Sigma_0$, $\text{Observ}(M_1, \Sigma_0)$, $\text{Observ}(M_2, \Sigma_0)$.

Block simulation is similar to block bisimulation which was defined in [15] for the purpose of checking correctness properties for parameterized distributed systems composed of similar processes connected in ring network. It is also close to visible simulation introduced in [3] and studied in [26]. Quasi-block simulation is an extension of block simulation. The necessity of this extension stems from the fact that asynchronous composition of LTSs (unlike synchronous one) is not monotonic w.r.t. block simulation.
3 The Basic Features of Quasi-block Simulation

Proposition 1. \( M_1 \preceq^{qb}_{\Sigma_0} M_2 \iff M_1 \preceq^{qb}_{\Sigma_0} M_2 \).

This statement follows immediately from the definitions above. Moreover, \(qb\)-simulation can be reduced to block simulation.

Consider two LTSs \( M_i = (S^i, S_0^i, A^i, R^i, \Sigma^i, L^i) \), \( i = 1, 2 \), such that \( M_1 \preceq^{qb}_{\Sigma_0} M_2 \) w.r.t. sets of events \( E^1 \) and \( E^2 \). Denote by \( \varepsilon \) an auxiliary visible action such that \( \varepsilon \notin A^1 \cup A^2 \), and build the LTSs \( \widetilde{M}_i = (S^i, S_0^i, A^i \cup \{ \varepsilon \}, \tilde{R}^i, \Sigma^i, L^i) \), \( i = 1, 2 \), such that \((s, a, t) \in \tilde{R}^i\) iff either \( a \neq \varepsilon \) and \((s, a, t) \in R^i\), or \( a = \varepsilon \) and \((s, \tau, t) \in E^i\). Thus, \( \varepsilon \) marks all those invisible transitions that are included in the sets of events \( E^1 \) and \( E^2 \).

Theorem 1. \( M_1 \preceq^{qb}_{\Sigma_0} M_2 \iff \widetilde{M}_1 \preceq^{qb}_{\Sigma_0} \widetilde{M}_2 \).

Theorem 1 may have a considerable utility in checking \(qb\)-simulation, since it provides a way of taking an advantage of efficient simulation-checking algorithms \([11, 17]\) that are applicable to block simulation. Quasi-block (unlike visible or block simulations) is preserved under asynchronous compositions of LTSs.

Theorem 2. Let \( M_i = (S^i, S_0^i, A^i, R^i, \Sigma^i, L^i) \), \( i = 1, 2, 3, 4 \), be four LTS’s such that \((\Sigma^1 \cup \Sigma^2) \cap (\Sigma^3 \cup \Sigma^4) = \emptyset\), \( A^1 = A^2 = A^4 = \emptyset\), \( A^3 = A^6 = A^6 = \emptyset\), and \( A^3 \cap A^6 = \emptyset\). Let \( \Sigma' \) and \( \Sigma'' \) be the distinguished sets such that \( \Sigma' \subseteq (\Sigma^1 \cup \Sigma^2) \) and \( \Sigma'' \subseteq (\Sigma^3 \cup \Sigma^4) \). Let \( \Pi = (\Delta, \nabla) \) be a synchronizer such that \( \Delta \subseteq A' \), and \( \nabla : \Delta \rightarrow A'' \). Then \( M_1 \preceq^{qb}_{\Sigma_0} M_2 \) and \( M_3 \preceq^{qb}_{\Sigma_1} M_4 \) implies \( M_1 \parallel M_3 \preceq^{qb}_{\Sigma_0 \cup \Sigma_1} M_2 \parallel M_4 \).

Yet another simulation which has close relationships with quasi-block one is stuttering simulation. It was introduced in \([3]\) and enjoys wide applications in the framework of partial order reduction technique (see \([9, 18]\)).

Let \( M_i = (S^i, S_0^i, A^i, R^i, \Sigma^i, L^i) \), \( i = 1, 2 \), be two LTSs, and \( \Sigma_0 \subseteq \Sigma^1 \cap \Sigma^2 \). A relation \( H \subseteq S^1 \times S^2 \) is called a stuttering simulation w.r.t. \( \Sigma_0 \) iff every pair \((s', s'') \in H\) complies with the following requirements:

1. \( L^1(s') \cap \Sigma_0 = L^2(s'') \cap \Sigma_0 \).
2. For every path \( \pi', \pi'' = s'_{1} \xrightarrow{a_1} s'_{2} \xrightarrow{a_2} \ldots \xrightarrow{a_{k-1}} s'_{k} \xrightarrow{a_k} s' \), \( s'_{1} \xrightarrow{a_1} s''_{1} \xrightarrow{a_2} \ldots \xrightarrow{a_{k-1}} s''_{k} \xrightarrow{a_k} s'' \) there is a path \( \pi', \pi'' = s''_{1} \xrightarrow{a_1} s''_{2} \xrightarrow{a_2} \ldots \xrightarrow{a_{k-1}} s''_{k} \xrightarrow{a_k} s'' \) and partitions \( P'_1P'_2 \ldots P'_m \) and \( P''_1P''_2 \ldots \) of \( \pi' \) and \( \pi'' \), such that for every \( i \geq 1 \) the sub-paths \( P'_i \) and \( P''_i \) match, i.e. \((s', s'') \in H\) holds for every pair of states \( s' \in P' \) and \( s'' \in P'' \).

We write \( M_1 \preceq^{st}_{\Sigma_0} M_2 \) to indicate the existence of stuttering simulation between \( M_1 \) and \( M_2 \).

Theorem 3. \( M_1 \preceq^{qb}_{\Sigma_0} M_2 \implies M_1 \preceq^{st}_{\Sigma_0} M_2 \).

A proof of this theorem is straightforward.

Since stuttering simulation preserves the satisfiability of temporal formulae from \(\text{ACTL}^*_{\land X}\), Theorem 3 brings us to the following conclusion.
Theorem 4. Suppose that a qb-simulation $M_1 \preceq_{qb}^\Sigma M_2$ is well-formed w.r.t. a $ACTL^*_X$-formula $\varphi$, and $M_2 \models \varphi$. Then $M_1 \models \varphi$ as well.

As it may be seen from the definition, stuttering simulation does not take into account any actions, but even in the case when $A_1 = A^2 = \emptyset$ it is weaker than qb-simulation (see Example 2 in Appendix 1). The fact that qb-simulation is stronger than stuttering simulation implies that the former is easy for checking and more feasible for practical applications in the framework of induction-based verification techniques.

4 Applying Quasi-block Simulation to the Verification of Asynchronous Networks

There are very few papers (in fact, the authors of [4] were not aware of any) where the induction-based verification technique is applied to asynchronous networks. In [15] parameterized systems composed of identical asynchronous processes which are arranged in a ring topology and communicate by passing a boolean token were considered. It has been shown that for several classes of indexed $CTL^*_X$ properties a cutoff effect takes place, i.e. the verification of the whole parameterized system can be reduced the model checking of finitely many instances of the system. In a certain sense, a cutoff plays a role of an invariant for such systems. In [10] the results of Emerson and Namjoshi were extended from rings to other classes of asynchronous networks. Nevertheless, many interesting classes of parameterized asynchronous systems do not fall into this category.

To the best of our knowledge the only paper where induction-based verification is applied to asynchronous networks is that of Clarke, Grumberg, and Jha [8]. In this paper they represented parameterized systems by network grammars, and used regular languages to express state properties. To generate invariants they developed an unfolding heuristics: given a parameterized system $\{P_k\}_{k=1}^\infty$ to find $n$ such that $h(P_{n+1}) \preceq h(P_n)$, where $\preceq$ is a strong simulation and $h$ is some appropriate abstraction. Much attention has been paid to the development of effective technique for constructing required abstractions.

To demonstrate that quasi-block simulation makes it possible to get rid of abstraction in induction based invariant generation we apply this approach to the verification of Resource ReSerVation Protocol (RSVP).

4.1 Resource Reservation Protocol

Resource reservation protocol (RSVP) [28] is designed to reserve resources in a network. The nodes of a network that hold these resources are called producers (or senders). Resource reservation procedure is launched by the consumers of resources (they are also called receivers). For instance, a consumer may reserve a download rate on the path from a producer to play video files stored on the producer without visible latency. A primary feature of RSVP is its scalability: RSVP scales to very large multicast groups because it uses receiver-orientated
reservation requests that merge as they progress up the multicast tree. The reservation for a single receiver does not need to travel to the source of a multicast tree (producer); it travels only until it reaches a reserved branch of the tree. While RSVP is designed for multicast applications, it may also make unicast data flows. RSVP is a unidirectional protocol. Resource reservation requests are sent upstream, from consumers to producers, whereas data is sent downstream, from producers to consumers. Any node in a network is allowed to be both a producer and a consumer. The protocol supports multiple reservation sessions, i.e. one consumer may reserve some resources in one session and other resources in another session. So, at the same time a host may be both a consumer in one session and a producer in another one. RSVP does not perform its own routing; instead it uses underlying routing protocols to determine where it should carry reservation requests. As routing changes paths to adapt to topology changes, RSVP adapts its reservation to the new paths wherever reservations are in place.

Producers and consumers communicate by sending messages. We will restrict our consideration to the messages of the following 8 types: path, resv, path_teardown, resv_teardown, path_refresh, resv_refresh, path_error, resv_error. A path message is sent downstream by a producer to allocate available communication paths, whereas resv messages are sent along such paths upstream by consumers as requests for reservation of resources. A path_teardown message is sent by a producer when it breaks sending data. A resv_teardown message is sent by a consumer when it abandons receiving data. From time to time path_refresh messages are sent downstream to make sure that communication paths are still active, and resv_refresh messages are sent upstream as reservation acknowledgments. A path_error message is sent upstream whenever an error occurs during path allocation. A resv_error message is sent downstream when a reservation error occurs.

The outline of communication in the protocol is as follows. Producers send path messages downstream along available routes that have been created by some routing protocol. In response to these messages consumers send upstream resv messages as reservation requests. Intermediate nodes (we will call them routers) check, whether reservation requests may be satisfied; if such is the case then they send reservation request upstream. The important feature of RSVP is that reservation requests are merged when sent upstream. When a reservation messages are delivered to a producer it sends data along the selected routes to the subscribed consumers. At any time a producer or a consumer may send a teardown message to cancel a communication session. The nodes refresh periodically the sessions by exchanging path_refr and resv_refr messages.

A model of RSVP. Some properties of RSVP have been already verified in [12, 30], S.J. Creese and J. Reed in [12] used CSP to construct a formal model of some fragment of RSVP to check the property of reservation merging in a binary multicast tree. In [30] a model of RSVP with one producer, one router and one consumer has been studied by means of Coloured Petri Nets; this approach extended substantially the set of properties that were verified. We do not study
RSVP in corpore either and restrict our consideration to some formal scale-down model of RSVP. The principal simplifications are as follows.

**Topology.** Following [12] we consider binary trees only. In this case only one producer per session is allowed. The routers and consumers form a binary tree where the routers are placed in the intermediate nodes and the consumers are placed in the leaves. The only producer is attached to the router in the root of the tree.

**Multicast vs. unicast.** When dealing with a unicast messaging one have to attach a destination (and possibly source) address to every message. In this case one should either to consider models with potentially infinite state space (as address size grows with the number of processes), or to consider models with a bounded number of consumers. In our model the producer uses multicast messages only. A router does not necessarily send messages to both descendants anyway.

**Reservation decisions.** In RSVP each intermediate node uses Admission Control to check, whether the node has sufficient available resources to supply the requested quality of service. In our current model it is assumed that the routers have an unbound amount of resources, which leads to a successful reservation on every request. We also take no account of Policy Control which is intended to determine whether the consumer has a permission to make the reservation.

**Failures.** We assume that all channels and processes are reliable: no messages could be lost and no routers could be stuck.

**Other abstractions.** When building our model we do not touch such issues as reservation confirmations, policy control, and timeouts.

The model of RSVP we deal with is a family of parameterized systems generated by the following network grammar $G$:

\[
\begin{align*}
P & \rightarrow p \parallel T \\
T & \rightarrow r \parallel T \parallel T \\
T & \rightarrow r \parallel c \parallel c
\end{align*}
\]

where terminals $p$ (producer), $c$ (consumer), and $r$ (router) are finite state processes. We denote by $L(G)$ the set of networks generated by the grammar $G$.

As $G$ generates only tree networks, it is more suitable to use traditional algebraic notation for writing network grammar terms. Thus, we will write $p(T)$, $r(T, T)$, $r(c, c)$ for the terms in the righthand side of the grammar rules. For instance, a term $p(r(r(c, c), r(c, c)))$ specifies a model composed of one producer, three routers, and four consumers; the network of this model is a tree of height 3. Sometimes a behaviour of a producer is of no importance for the analysis of a model. In these cases we restrict our consideration to tree networks derived from non-terminal $T$ only. Such networks are specified by the terms of the form $r(t_1, t_2)$. In what follows when studying $qb$-simulation $r(t_1, t_2) \preceq_{\Sigma(r)} r(t'_1, t'_2)$
between reduced models \( r(t_1, t_2) \) and \( r(t'_1, t'_2) \) we will assume that \( \Sigma(r) \) is the set of atomic propositions of the topmost router \( r \). Similarly, we assume that \( \Sigma(p) \) is the set of atomic propositions of the producer \( p \).

The models of RSVP were implemented in the model description language Promela [27] in the framework of the open-source software verification tool Spin [29]. The usage of Promela gave us an advantage of exploiting a simulator and model checker from Spin.

### 4.2 Finding Invariants

We apply the induction technique to verify the infinite family of parameterized finite state systems generated by the network grammar \( G \). The key point of our approach is that of finding an invariant of \( L(G) \). We made an attempt to find such invariant by analyzing the networks derived from non-terminal symbol \( T \).

Denote by \( T_N \) the set of tree networks of the height \( N \) derived from \( T \).

**Proposition 2.** If the following qb-simulations are valid

\[
\begin{align*}
& r(r(c, c), c) \preceq_{\Sigma(r)}^q r(c, r(c, c)) \quad (1) \\
& r(r(c, c), r(c, c)) \preceq_{\Sigma(r)}^q r(c, r(c, c)) \quad (2) \\
& r(c, r(c, r(c, c))) \preceq_{\Sigma(r)}^q r(r(c, c), r(c, c)) \quad (3) \\
& r(r(c, r(c, c)), c) \preceq_{\Sigma(r)}^q r(r(c, c), r(c, c)) \quad (4) \\
& r(r(c, r(c, c)), r(c, r(c, c))) \preceq_{\Sigma(r)}^q r(r(c, c), r(c, c)) \quad (5)
\end{align*}
\]

then any network \( t \) from \( T_3 \) is qb-simulated by the model \( r(r(c, c), r(c, c)) \), i.e. \( t \preceq_{\Sigma(r)}^q r(r(c, c), r(c, c)) \).

**Proof.** Assumption (1) allows us to rotate a tree. Applying assumptions (1) and (2) to the subtrees of a network from \( T_3 \) we conclude that for any \( t \in T_3 \) there exists such a \( t' \subseteq \{ r(c, r(c, r(c, c))), r(r(c, r(c, c)), c), r(r(c, r(c, c)), r(c, r(c, c))) \} \) that \( t \preceq_{\Sigma(r)}^q t' \). In view of the assumptions (3), (4) and (5) the latter is reduced to qb-simulation \( t \preceq_{\Sigma(r)}^q r(r(c, c), r(c, c)) \). All these reductions can be made due to the monotonicity Theorem 2. This completes the proof. \( \square \)

As it may be seen from Proposition 2 the model \( Inv = r(r(c, c), r(c, c)) \) is a keystone of our construction.

**Proposition 3.** If the assumptions (1)–(5) are valid, then the model \( Inv \) qb-simulates every network \( t \) from \( T_N \), \( N \geq 3 \), i.e. \( t \preceq_{\Sigma(r)}^q Inv \).

**Proof.** By induction on the height \( N \) of tree networks. By Proposition 2, if qb-simulations (1)–(5) are valid then \( t \preceq_{\Sigma(r)}^q Inv \) holds for every network \( t \) from \( T_3 \). Suppose that \( Inv \) qb-simulates every network from \( T_N \). Let \( t \) be an arbitrary network \( T_{N+1} \). Consider a tree network \( t' \) which is obtained from \( t \) by
substituting the network Inv instead of every subtree of height 3 in $t$. Clearly, the height of $t'$ is $N$. Furthermore, by Theorem 2, if Inv qb-simulates every network from $T_3$ then $t \preceq_{\Sigma(r)}^{qb} t'$. Hence, $t \preceq_{\Sigma(r)}^{qb} \text{Inv}$. \hfill $\square$

By combining Theorem 2 and Proposition 3, we arrive at the following conclusion.

**Corollary 1.** If the assumptions (1)–(5) are valid then the model $p(\text{Inv})$ is an invariant of the family of models generated by the network grammar $G$, i.e. $p(t) \preceq_{\Sigma(r)}^{qb} p(\text{Inv})$ holds for every network $t$ of height $N, N \geq 3$, derived from non-terminal $T$.

Thus, to certify that the finite model $p(\text{Inv})$ is an invariant of the infinite family of models $L(G)$ it is sufficient to make certain that the assumptions (1)–(5) are true.

### 4.3 Checking Quasi-block Simulation

When checking quasi-block simulation between finite models we rely on Proposition 1. Since $M_1 \preceq_{\Sigma}^{\Sigma} M_2$ implies $M_1 \preceq_{\Sigma}^{qb} M_2$, we reduce quasi-block simulation checking of the assumptions (1)–(5) to block simulation checking of the corresponding models.

To check that a pair of states $(s, u)$ in some LTSs $M_1, M_2$ complies with the definition of block simulation one should check each pair in every finite block outgoing from the states $s$ and $u$. To reduce the complexity of checking procedure we introduce a new concept of semi-block simulation. Let $M_i = (S^i, S_0^i, A^i, R^i, \Sigma^i, L^i), i = 1, 2$, be two LTSs. Let $\Sigma_0$ be a subset of $\Sigma^1 \cap \Sigma^2$, and $E^1$ and $E^2$ be some sets of events of $M_1$ and $M_2$.

**Definition 5.** A binary relation $H \subseteq S^1 \times S^2$ is called a semi-block simulation on $M_1$ and $M_2$ w.r.t. $\Sigma_0$, $E^1$, $E^2$, iff every pair $(s_1, t_1) \in H$ meets the following requirements:

1. $L^1(s_1) \cap \Sigma_0 = L^2(t_1) \cap \Sigma_0$,
2. For every finite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1} \in \text{MAXF}(E^1, s_1)$ there is a block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{a} t_{n+1} \in \text{MAXF}(E^2, t_1)$ such that:
   - (a) if $n > 1$ then $(s_1, t_n) \in H$ and $(s_{m+1}, t_{n+1}) \in H$;
   - (b) if $n = 1$ then $(s_{m+1}, t_{n+1}) \in H$;
3. (divergence) if a $\tau$-cycle is reachable from state $s_1$ by some $\tau$-path, i.e. $\text{MAXF}(E_1, s_1) \neq \emptyset$, then there exists such a $\tau$-cycle $\gamma$ reachable from state $t_1$ by some $\tau$-path and a state $t_n \in \gamma$ that: $(s_1, t_n) \in H$.

The items of this definition are illustrated in Fig. 4.3. We write $M_1 \preceq_{\Sigma_0}^{\Sigma} M_2$ iff there exist two sets of events $E^1$ and $E^2$ of LTSs $M_1$ and $M_2$ and a binary relation $H \subseteq S^1 \times S^2$ such that $H$ is a semi-block simulation on $M_1$ and $M_2$ w.r.t. $\Sigma_0$, $E^1$, $E^2$, and for every initial state $s_0 \in S^1_0$ there exists an initial state $t_0 \in S^2_0$ such that $(s_0, t_0) \in H$. 

Theorem 5. $M_1 \preceq \Sigma M_2 \iff M_1 \preceq \Sigma M_2$.

We implemented a naive algorithm for checking semi-block simulation which works iteratively according to the definition above. It adds all pairs with the same labelling of states to the relation and then removes those pairs that do not satisfy the definition. However, in this case at the first iteration the relation may have a very large size close to $|M_1| \times |M_2|$. To avoid such effect we use a lazy approach. All pairs in the relation are partitioned into two groups: positive pairs (that satisfy the definition) and negative pairs (that do not satisfy it). The computation of a semi-block simulation begins with considering the initial pairs as positives. Next, the positive pairs are checked one by one following the definition. The computation continues while some pairs are brought into consideration or change their status from positive to negative. Since the set of negative pairs grows monotonically the computation will eventually terminate.

A semi-block simulation checking algorithm has been implemented in C++ using `std::map` to store relation. Our simulation-checking tool built the required block simulations that validate the assumptions (1)–(5). When checking $r(r(c, r(c, c)), r(c, r(c, c)))$ versus $Inv$ we had ran out of memory (2 Gigabytes). To bypass the memory limitation we implemented a relation storage as a minimized automaton from Spin (see [25]). We found that this not only drastically decreases memory consumption, but also speed-up the computation.

The results of our experiments are depicted in Table 1. The program was running on 2.4 GHz AMD Opteron provided by Laboratory of Computer Systems, Moscow State University.

The checking of $r(r(c, r(c, c)), r(c, r(c, c)))$ vs $Inv$ takes a considerable amount of time. As checking of each pair is performed proportionally fast and computation slows up after about 50 millions of positives added, we guess two reasons of such a degradation. First, we used state enumeration in minimized DFA, which looks to be less efficient than insertion, deletion and lookup operations. Second, minimized DFA hash tables and splay trees should be tuned up on large problems.
Table 1. Validating the assumptions (1)–(5). Results.

| Assumption on $\mathcal{L}(G)$ against any $\text{ACTL}_X$-specification $\varphi$ is it sufficient to demonstrate that the invariant $p(r(r(c,c),r(c,c)))$ and networks $p(r(c,r(c,c)))$, $p(r(r(c,c),c))$, $p(r(c,c))$, whose height is less than 3, satisfy $\varphi$.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>#States of $M_1$</th>
<th>#States of $M_2$</th>
<th>#Positive pairs</th>
<th>#Negative pairs</th>
<th>Time</th>
<th>Memory</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>1732</td>
<td>1277</td>
<td>15902</td>
<td>905</td>
<td>2 sec</td>
<td>22M</td>
<td>yes</td>
</tr>
<tr>
<td>(2)</td>
<td>24993</td>
<td>1277</td>
<td>223304</td>
<td>4337</td>
<td>31 sec</td>
<td>39M</td>
<td>yes</td>
</tr>
<tr>
<td>(3)</td>
<td>14672</td>
<td>24993</td>
<td>1425766</td>
<td>5443</td>
<td>7 min</td>
<td>43M</td>
<td>yes</td>
</tr>
<tr>
<td>(4)</td>
<td>21659</td>
<td>24993</td>
<td>$3.8 \times 10^6$</td>
<td>9002</td>
<td>19 min 30 sec</td>
<td>44M</td>
<td>yes</td>
</tr>
<tr>
<td>(5)</td>
<td>$3.8 \times 10^6$</td>
<td>24993</td>
<td>$3.5 \times 10^6$</td>
<td>116172</td>
<td>76 hours</td>
<td>49M</td>
<td>yes</td>
</tr>
</tbody>
</table>

4.4 Using Invariants for Checking Properties of RSVP.

As it follows from Theorem 4 and Corollary 1, to check the whole infinite family of networks $\mathcal{L}(G)$ against any $\text{ACTL}_X$-specification $\varphi$ it is sufficient to demonstrate that the invariant $p(r(r(c,c),r(c,c)))$ and networks $p(r(c,r(c,c)))$, $p(r(r(c,c),c))$, $p(r(c,c))$, whose height is less than 3, satisfy $\varphi$.

Our main efforts were focused on the computation of block simulation and thus finding a desired invariant. So, we have not checked a lot of properties. One property we checked on the model $p(r(r(c,c),r(c,c)))$ (one producer, three routers, and four consumers) is the requirement that the consumer does not receive path teardown acknowledge while it does not send path teardown message.

This specification is expressed in Linear Temporal Logic as:

$$\varphi_1 = G \neg \text{producer}!\text{path.tear} \rightarrow G \neg \text{producer}?\text{path.tear.acknowledge}$$  \hfill (6)

The second checked property is the property of reservation merging, which had been checked in [12]. In terms of our model it is written as an absence of reservation request while router is in “reserved” state:

$$\varphi_2 = G (\text{router1.reserved} \rightarrow \neg \text{router1.parent!resv})$$  \hfill (7)

By applying Spin model-checker we found that specification $\varphi_1$ is satisfied on $p(r(r(c,c),r(c,c)))$ and $\varphi_2$ is satisfied on $r(r(c,c),r(c,c))$. Due to Proposition 3 and Corollary 1 we conclude that properties are satisfied on any tree of length greater than 1:

$$p(w) \models \varphi_1, \text{ where } w \in \bigcup_{i=2}^{\infty} T_i$$  \hfill (8)

$$w \models \varphi_2, \text{ where } w \in \bigcup_{i=2}^{\infty} T_i$$  \hfill (9)
Conclusions and Directions for Future Research

There is a number of tasks to be solved next to make a good "reputation" for quasi-block simulation. Certainly, we have to find out some other practical case studies that could indicate convincingly the advisability of using quasi-block simulation in the verification of parameterized systems. Thus, we are about to extend our condensed version of RSVP by introducing to the model bounds on the resources and adding some elements from Admission Control module. We are also interested in checking fault tolerance properties under the assumption that some routers in communications paths may fail. This line of investigation depends to a large extent also on how much effectively quasi-block simulation can be checked. We assume that Theorems 1, 5 could give an essential prerequisite for constructing efficient checking procedures. So far we use an explicit representation of simulation relations (with a minor usage of symbolic techniques for storing them). We think that further improvements may be achieved with the application of a game-theoretic approach in conjunction with symbolic computations. Of some interest is also a search of some parameterized system which can be served as indicative counter-example to reveal the limitations of quasi-block simulations and to outline the ways for its further improvement.

References


