Journal of Symbolic Computation 45 (2010) 1144–1162

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

An invariant-based approach to the verification of asynchronous parameterized networks☆

Igor V. Konnov [1], Vladimir A. Zakharov

Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Moscow, RU-119991, Russia

ARTICLE INFO

Article history:
Received 7 July 2008
Accepted 10 November 2008
Available online 9 March 2009

Keywords:
Program verification
Asynchronous networks
Invariant generation
Induction
Simulation
Model checking

ABSTRACT

A uniform *verification problem for parameterized systems* is to determine whether a temporal property is satisfied 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 system. At the second stage, as soon as an invariant of the parameterized system is obtained, we verify it by means of a conventional model checking tool for temporal logics. An invariant system is one that is greater (with respect to some preorder relation) than any instance of the parameterized system. Therefore, the preorder relation involved in the *invariant rule* is of considerable importance. For this purpose we introduce a new type of simulation preorder — quasi-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 in the induction-based verification techniques for asynchronous networks. To demonstrate the feasibility of quasi-block simulation we implemented this technique and applied it to the verification of the Resource ReSerVation Protocol (RSVP).

☆ This research was supported by INTAS grant 05-1000008-8144 and RFBR grant 06-01-00106.
E-mail addresses: konnov@cs.msu.su (I.V. Konnov), zakh@cs.msu.su (V.A. Zakharov).
[1] Tel.: +7 495 939 4671; fax: +7 495 939 2596.

0747-7171/$ – see front matter
doi:10.1016/j.jsc.2008.11.006

1. Introduction

Verification plays an important role in designing reliable computer systems. Two main approaches to program verification are testing (Kaner et al., 1999) and formal verification (Apt and Olderog, 1997; Clarke et al., 1999). The aim of testing is to detect program failures. It helps to uncover and correct defects of a program behavior but it cannot establish, except for trivial cases, that a program functions properly. 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 (see, for example, Clarke et al. (1993), Chehaibar et al. (1996) and Cimatti (2000)).

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 a transition system derived from hardware or software design P) satisfies a formal specification φ (usually represented as a temporal logic formula). When $M(P)$ is a finite state model then one could find a rich variety of model checking procedures (see Clarke et al. (1999)). In what follows we will assume that each system (process) P under consideration has only finite state set and will not distinguish P 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 $p_1 \| p_2 \| \ldots \| p_k \| q$ of similar processes $p_1, p_2, \ldots, p_k$ (which are instances of a prototype process p) and some distinguished process q ("environment"). Then the uniform verification problem for parameterized systems is formulated as follows: given an infinite family $\mathcal{F}$ of systems $P_k = p_1 \| p_2 \| \ldots \| p_k \| q$ and a temporal formula φ, check that each transition system $M(P_k)$ satisfies φ.

Though Apt and Kozen (1986) proved that the problem is undecidable in general, some positive results have been obtained for specific parameterized systems. For the most part four basic techniques, namely, *symmetry*, *cut-off*, *abstraction*, and *induction* are employed to extend the applicability of conventional model checking to restricted families of parameterized systems.

Symmetry

Symmetry is commonly used to avoid the combinatorial explosion in the number of states in transition systems (see Clarke et al. (1999)). Suppose that a transition system P has a non-trivial group G of permutations on the state set S; each permutation from G preserves both the transition relation R and the labelling function L. Then one could replace the system P with a quotient model P_G, where the state space $S_G = \{\theta(s) \,:\, s \in S, \theta(s) = \{r \,:\, \exists \sigma \in G\, (\sigma(s) = r)\}\}$ is the set of orbits of the states in S. If the property φ to be checked is invariant under G then $P, s \models \varphi \iff P_G, \theta(s) \models \varphi$. It is obvious that symmetry-based reduction does the job when a network topology is highly symmetric, i.e. it is a ring, clique, hypercube, etc. The idea of exploiting symmetry for state set reduction was introduced by Clarke and Filkorn (1993), Emerson and Namjoshi (1995) and Emerson and Sistla (1996). Symmetry-based reduction has been successfully applied to a number of case studies (see Calder and Miller (2002) for a survey) and now it is implemented within a framework of many model checkers (see Ip and Dill (1997) and Bosnacki et al. (2001)). However, in many practical cases this approach runs into obstacles, since the problem of finding orbit representatives is as hard as graph isomorphism problem. Manku et al. (1998) and Donaldson and Miller (2005) have demonstrated a considerable progress in automatic symmetry detection, but this problem still remains the main critical point of the symmetry-based reduction techniques.

Emerson and Namjoshi (1995) demonstrated how to exploit symmetry to cope with model checking of parameterized specifications. Suppose that a system under consideration is a token ring $Ring(n) = p_1 \| p_2 \| \ldots \| p_{n-1} \| p_n$, where p_n is the initial token distribution process, and a specification to be checked is a CTL*-X formula $\varphi_n = \bigwedge_{i=1}^{n} \psi[i]$, where $\psi[i]$ refers only to the events occurred in the process p_i. Then $Ring(n)$ satisfies φ_n iff $Ring(2)$ satisfies φ_2. Similar effect has been revealed for parameterized specifications of some other forms.

Cut-offs

A successful application of symmetry for the verification of some parameterized networks gave impetus to the development of cut-off technique. Its principal idea is as follows. Let $\approx$ be an equivalence relation on transition systems which preserves a given set of formulae *Form*, i.e. if $M_1 \approx M_2$ then $M_1 \models \varphi \iff M_2 \models \varphi$ holds for every formula φ from *Form*. Then to check a specification φ against a parameterized system $\mathcal{F} = \{P_k\}_{k=1}^{\infty}$ one needs only to find such N (a *cut-off number*) that $P_{N+i} \approx P_N$ holds for every i, $i \geq 0$, and check φ on a finite set of finite state models $P_1, P_2, \ldots, P_N$. The system P_N distinguished thus is called a *cut-off process*.

A cut-off technique was introduced by Emerson and Namjoshi (1995). They verified a family of unidirectional token rings and used block bisimulation for the equivalence relation $\approx$. Cut-off approach was successfully implemented by Emerson and Kahlon (2003) and Emerson and Kahlon (2004) and applied to the verification of bidirectional token rings, initialized broadcast protocols, and guarded broadcast protocols including cache coherence protocols for multiprocessor computers.

Abstraction

Abstraction is likely to be the most important technique for coping with state explosion problem in model checking. It provides a way to replace a large model P with a smaller one $h(P)$ such that $h(P)$ inherits some properties of P. This may be achieved by picking out a distinguished set of formulae A and introducing an equivalence relation over the state set S such that if two states are equivalent then for every formula ψ from A they both either satisfy or falsify ψ. Using the equivalence classes as abstract states and defining an abstract transition relation appropriately, one gets an abstract model $h(P)$. If a temporal formula φ to be checked is built of formulae from the set A then $h(P) \models \varphi \Longrightarrow P \models \varphi$.

A theoretical framework for abstraction techniques has been developed by Clarke et al. (1992), Dams et al. (1994), Pardo (1997) and Kesten and Pnueli (2000). Abstraction also gives rise to a new verification paradigm — counterexample guided abstraction refinement (see Clarke et al. (1999) and Henzinger et al. (1999)).

Abstraction has been widely applied in verification of parameterized systems. With the essential help of abstraction it does become possible to apply model checking to verify infinite families of models (see German and Sistla (1992), Clarke et al. (1995), Lesens and Saidi (1997), Ip and Dill (1997) and Clarke et al. (2004)). Unfortunately, the advanced abstraction techniques for parameterized model checking require user assistance in providing key elements and mappings. Therefore, in most cases it gives effect only when it is coupled with other techniques that are more suitable for automatic application.

Induction

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

1. the composition operator $\|$ is monotonic w.r.t. $\preceq$, i.e. $P_1 \preceq P_1'$ and $P_2 \preceq P_2'$ imply $P_1 \| P_2 \preceq P_1' \| P_2'$;
2. the preorder $\preceq$ preserves the satisfiability of formulae φ from *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 $P_k = p_1 \| p_2 \| \ldots \| p_k \| q$, find a finite transition system $\mathcal{A}$ such that

3. $P_n \preceq \mathcal{I}$ for some n, $n \geq 1$;
4. $p \| \mathcal{I} \preceq \mathcal{I}$.

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 specification to be checked is expressed by a formula φ 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 $\mathcal{I}$ and P_k, $1 \leq k < n$, against φ. The latter may be done by means of any suitable model checking technique for finite state transition systems.

This approach to the verification of parameterized networks was introduced by Kurshan and McMillan (1989) and Wolper and Lovinfosse (1989) and developed in many papers (see Calder and Miller (2002) for a survey). The central problem here is that of deriving a general method for constructing invariants. In many cases invariants can be obtained by using 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 idea suggested by Clarke et al. (1995, 1997) opens new ways for developing fully automated approach for verifying parameterized networks. A typical verification scenario looks as follows.

(1) Given a parameterized system $\{P_k\}_{k=1}^{\infty}$ and a property φ_k, a symmetry-based reduction technique is used to degenerate φ_k to a more simple formula ψ.
(2) The formula ψ is used to generate an abstraction h such that $h(h(P_i) \| h(p_{i+1})) \preceq h(P_{i+1})$ holds for every i, $i \geq 1$.
(3) Next an attempt is made to find n such that $h(P_{n+1}) \preceq h(P_n)$.
(4) As soon as such n is found a model checking technique is applied to verify the satisfiability of ψ on the finite transition systems $h(P_n)$ and P_k, $1 \leq k < n$.

As it may be seen from this description the right choice of abstraction h and a preorder $\preceq$ is of prime importance for the successful application of this heuristic in practice. Clarke et al. (1995, 1997) demonstrated how to derive an appropriate abstraction h from the property ψ to be checked when the latter is represented by a finite automaton. Less attention has been paid to $\preceq$. In the papers by Clarke et al. (1995) and Emerson and Namjoshi (1995) strong simulation and block bisimulation respectively were used as a preorder $\preceq$, but as 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 the preorder $\preceq$ is, the larger in number are the cases of parameterized systems for which this approach succeeds. Furthermore, a careful choice of $\preceq$ makes it possible to circumvent difficulties pertaining to abstractions: if $\preceq$ is loose enough an invariant P_n can be found without resorting to abstraction h. To be certain that this effect could appear we introduced (see Konnov and Zakharov (2005)) a block simulation preorder (which is an amalgamation of block bisimulation suggested by Emerson and Namjoshi (1995) and visible simulation proposed by Penczek et al. (1999)) and applied this preorder to generate straightforwardly invariants of some parameterized systems.

In this paper we continue our 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 the verification of the Resource ReSerVation Protocol (RSVP). Hitherto formal verification of RSVP has been studied by Creese and Reed (1999) with the help of CSP and by Villapol (2003) 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. Some strategies that alleviate the generation of invariants for tree-like networks are developed Section 4. In Section 5 we consider RSVP as a case study and demonstrate how to verify this protocol

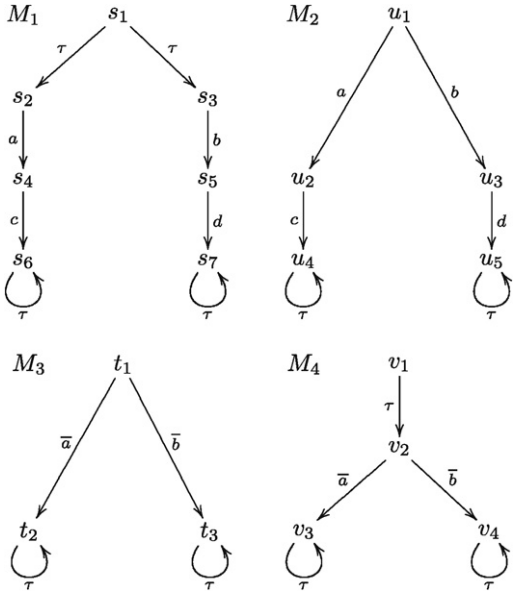


Fig. 1. Examples of LTSs.

using induction technique based on quasi-block simulation for generating a suitable invariant. Section 6 concludes with directions for future research.

2. Definitions

Definition 1. A *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 τ),
- $R \subseteq S \times (A \cup \{\tau\}) \times S$ is a labelled transition relation,
- Σ is a nonempty set of atomic propositions,
- $L : S \to 2^{\Sigma}$ is an evaluation function on the set of states. □

Examples of LTSs are depicted in Fig. 1. 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* π of LTS M is a finite or infinite sequence $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{j-1}} s_j \xrightarrow{a_j} \cdots$ of transitions $s_i \xrightarrow{a_i} s_{i+1}$.

Temporal specifications (or properties) of parameterized systems are expressed in temporal logics. The logic used in the framework of induction-based verification techniques are usually the Full Branching Time Logic CTL* or its sub-logics ACTL* and ACTL*-X. An important factor in deciding between them is a capability of a preorder $\preceq$ used in the verification procedure to preserve the satisfiability of temporal formulae. We do not define the syntax and the semantics of these logics; they can be found in many textbooks, e.g. in Clarke et al. (1999).

Let $M_1 = \langle S^1, S_0^1, A^1, R^1, \Sigma^1, L^1\rangle, M_2 = \langle S^2, S_0^2, A^2, R^2, \Sigma^2, L^2\rangle$ be two LTSs such that $\Sigma^1 \cap \Sigma^2 = \emptyset$. We call any pair $\Gamma = \langle \Delta, {}^{-}\rangle$ a *synchronizer*, where $\Delta \subseteq A^1$, and ${}^{-} : \Delta \to A^2$ is an injection which binds some actions of M_1 and M_2. We write $\overline{\Delta}$ for the set $\{b \in A^2 \mid \exists a \in \Delta : \overline{a} = b\}$. When introducing a synchronizer we assume that a certain action a from Δ is executed only synchronously with its co-action $\overline{a}$. Thus, a pair $(a, \overline{a})$ forms a channel for communication between M_1 and M_2. One of these actions (say, a) may be thought as an action of sending a message, whereas the other (co-action $\overline{a}$) is an action of receiving a message.

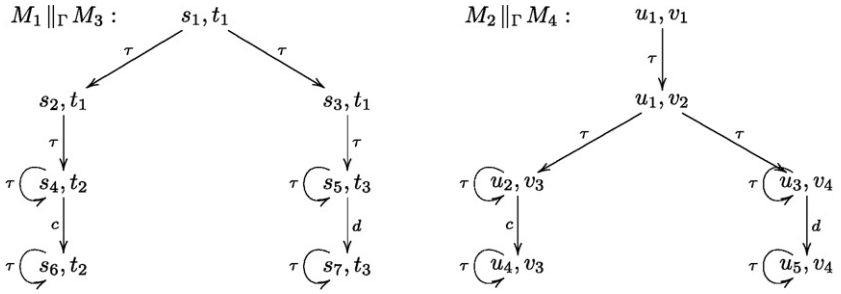


Fig. 2. Examples of parallel compositions of LTSs.

We define the operation for parallel composition of processes presented by LTSs, based on the interleaving paradigm: any two actions cannot happen at the same time.

Definition 2. The *(asynchronous) parallel composition* of LTSs M_1 and M_2 w.r.t. synchronizer Γ is an LTS $M = M_1 \|_\Gamma M_2 = \langle S, S_0, A, R, \Sigma, L\rangle$ such that

- $S = S^1 \times S^2$,
- $S_0 = S_0^1 \times S_0^2$,
- $A = (A^1 \cup A^2) \setminus (\Delta \cup \overline{\Delta})$,
- $\Sigma = \Sigma^1 \cup \Sigma^2$,
- $L((s,u)) = L^1(s) \cup L^2(u)$,
- 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 iff one of the following requirements is met:
  . $a \in A^1 \setminus \Delta, u = v, (s,a,t) \in R^1$
  (in this case we say that M_1 moves at the step $(s,u) \xrightarrow{a} (t,v)$),
  . $a \in A^2 \setminus \overline{\Delta}, s = t, (u,a,v) \in R^2$
  (in this case we say that M_1 moves at the step $(s,u) \xrightarrow{a} (t,v)$),
  . $a = \tau$, and there exists $b \in \Delta$ such that $(s,b,t) \in R^1$ and $(u,\overline{b},v) \in R^2$
  (in this case we say that M_1 and M_2 communicate at the step $(s,u) \xrightarrow{\tau} (t,v)$). □

Examples of some LTSs M_1, M_2, M_3, M_4 are depicted in Fig. 1. Their asynchronous parallel compositions $M_1 \| M_3$ and $M_2 \| M_4$ w.r.t. a synchronizer $\Gamma = (\{\{a,b\}, \{a \to \overline{a}, b \to \overline{b}\}\})$ are depicted in Fig. 2.

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

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

On the other hand, one may also distinguish some sets 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 $Observ(M, \Sigma_\varphi) \subseteq E$ then the set of events E will be called *well-formed* w.r.t. φ.

Definition 3. Let $M = M' \|_\Gamma M''$ be a parallel composition of some LTSs M', M'' and $\delta = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{i-1}} s_i \xrightarrow{a_i} \cdots$ be a path in LTS M. A *projection* $pr_{M'}(\delta)$ of δ on the model M' is defined by induction on the length n of δ as follows:

- if $\delta = (s'_1, s''_1)$ then $pr_{M'}(\delta) = s'_1$,
- if $\delta = \sigma \circ ((s'_n, s''_n) \xrightarrow{a}_M (s'_{n+1}, s''_{n+1}))$ then:
  . if M' does not move at the step $(s'_n, s''_n) \xrightarrow{a}_M (s'_{n+1}, s''_{n+1})$ then $pr_{M'}(\delta) = pr_{M'}(\sigma)$;
  . if M' moves at the step $(s'_n, s''_n) \xrightarrow{a}_M (s'_{n+1}, s''_{n+1})$ then $pr_{M'}(\delta) = pr_{M'}(\sigma) \circ (s'_n \xrightarrow{a} s'_{n+1})$;
  . if M' and M'' communicate at the step $(s'_n, s''_n) \xrightarrow{\tau}_M (s'_{n+1}, s''_{n+1})$ via a pair of synchronous actions $b, \overline{b}$ then $pr_{M'}(\delta) = pr_{M'}(\sigma) \circ (s'_n \xrightarrow{b} s'_{n+1})$. □

Definition 4. A *finite block* outgoing 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 $(s_i, \tau, s_{i+1}) \notin E$ for all $i : 1 \le i < m$. An *infinite block* outgoing from a state s_1 is an infinite sequence $B = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{\tau} \cdots$ of τ-transitions such that $(s_i, \tau, s_{i+1}) \notin E$ for all $i \ge 1$. □

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

Definition 5. Let $M_1 = \langle S^1, S_0^1, A^1, R^1, \Sigma^1, L^1 \rangle$, $M_2 = \langle S^2, S_0^2, A^2, R^2, \Sigma^2, L^2 \rangle$ be two LTSs. Let Σ be a subset of $\Sigma^1 \cap \Sigma^2$, and let E^1 and E^2 be subsets 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 (qb-simulation)* on M_1 and M_2 w.r.t. Σ, E^1, E^2, iff any pair $(s_1, t_1) \in H$ meets the following requirements:

QB1: $L^1(s_1) \cap \Sigma = L^2(t_1) \cap \Sigma$,

QB2: For every finite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1} \in MAXF(E^1, s_1)$ there is a block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{a} t_{n+1} \in MAXF(E^2, t_1)$ such that $(s_{m+1}, t_{n+1}) \in H$, and $(s_i, t_j) \in H$ holds for every pair i, j, $1 \le i \le m$, $1 \le j \le n$.

QB3: For every infinite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{\tau} \cdots \in MAXI(E^1, s_1)$ there is an infinite block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{\tau} \cdots \in MAXI(E^2, t_1)$, such that $(s_i, t_j) \in H$ holds for every pair i, j, $1 \le i, 1 \le j$. □

In what follows we will say that a block B'' referred to in the requirements QB2 and QB3 matches a block B' w.r.t. H. We write $M_1 \preceq^{qb}_{\Sigma} M_2$ 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 qb-simulation on M_1 and M_2 w.r.t. Σ, 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. φ then we say that the qb-simulation $M_1 \preceq^{qb}_{\Sigma_\varphi} M_2$ is also well-formed w.r.t. φ. A *block simulation* ($M_1 \preceq^{b}_{\Sigma} M_2$ in symbols) is a qb-simulation w.r.t. Σ, $Observ(M_1, \Sigma)$, $Observ(M_2, \Sigma)$.

Our notion of block simulation is similar to the notion of block bisimulation which was defined by Emerson and Namjoshi (1995) for the purpose of checking correctness properties for parameterized distributed systems composed of similar processes connected in a ring network. It is also close to visible simulation introduced by Browne et al. (1988) and Penczek et al. (1999). 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. The following example reveals this effect.

Example 6. Consider LTSs M_1, M_2, M_3, M_4 depicted in Fig. 1 and their parallel compositions $M_{13} = M_1 \parallel_\Gamma M_3$ and $M_{24} = M_2 \parallel_\Gamma M_4$ w.r.t. a synchronizer $\Gamma = (\{\{a, b\}, \{a \to \bar{a}, b \to \bar{b}\}\})$ depicted in Fig. 2.

As can be seen from the definition of block simulation, $M_1 \preceq^{b}_{\emptyset} M_2$ and $M_3 \preceq^{b}_{\emptyset} M_4$. To ascertain that there exists no block simulation for the compositions M_{13} and M_{24} consider a finite block

$$B' = (s_1, t_1) \xrightarrow{\tau} (s_2, t_1) \xrightarrow{\tau} (s_4, t_2) \xrightarrow{c} (s_6, t_2)$$

in the LTS M_{13}. The only blocks in M_{24} that may match B' are that of the form

$$B'' = (u_1, v_1) \xrightarrow{\tau} (u_1, v_2) \xrightarrow{\tau} (u_2, v_3) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (u_2, v_3) \xrightarrow{c} (u_4, v_3).$$

In this case the state (s_1, t_1) in B' must correspond to the state (u_2, v_3) in B''. But this is impossible, since the set $MAXF(Observ(M_{13}, \emptyset), (s_1, t_1))$ contains a finite block which ends with a transition $(s_5, t_3) \xrightarrow{d} (s_7, t_3)$, whereas all finite blocks from the set $MAXF(Observ(M_{24}, \emptyset), (u_2, v_3))$ end with a transition $(u_2, v_3) \xrightarrow{c} (u_4, v_3)$.

On the other hand, if we mark all communication transitions in both models M_{13} and M_{24} (namely, $(s_2, t_1) \xrightarrow{\tau} (s_4, t_2)$, $(s_3, t_1) \xrightarrow{\tau} (s_5, t_3)$, $(u_1, v_2) \xrightarrow{\tau} (u_2, v_3)$, and $(u_1, v_2) \xrightarrow{\tau} (u_3, \underset{\sim}{v_4})$) with the same distinguished visible action ε instead of τ then we obtain a pair of LTSs $\widetilde{M}_{13}$ and $\widetilde{M}_{24}$ such that $\widetilde{M}_{13} \preceq^{b}_{\emptyset} \widetilde{M}_{24}$. In fact, the latter implies $M_{13} \preceq^{qb}_{\emptyset} M_{24}$. □

3. The basic features of quasi-block simulation

Proposition 7. $M_1 \preceq^{b}_{\Sigma} M_2 \Longrightarrow M_1 \preceq^{qb}_{\Sigma} M_2$.

This statement follows immediately from the definitions above. Strange though it may seem, qb-simulation can be reduced to block simulation.

Consider two LTSs $M_1 = \langle S^1, S^1_0, A^1, R^1, \Sigma^1, L^1\rangle$ and $M_2 = \langle S^2, S^2_0, A^2, R^2, \Sigma^2, L^2\rangle$ such that $M_1 \preceq^{qb}_{\Sigma} M_2$ w.r.t. sets of events E^1 and E^2. Denote by ε an auxiliary visible action such that $\varepsilon \notin A^1 \cup A^2$, and build the LTSs $\widetilde{M}_i = \langle S^i, S^i_0, A^i \cup \{\varepsilon\}, \widetilde{R}^i, \Sigma^i, L^i\rangle, i = 1, 2$, such that $(s, a, t) \in \widetilde{R}^i$ iff either $a \neq \varepsilon$ and $(s, a, t) \in R^i$, or $a = \varepsilon$ and $(s, \tau, t) \in E^i$. Thus, ε marks all those invisible transitions that are included in the sets of events E^1 and E^2.

Proposition 8. $M_1 \preceq^{qb}_{\Sigma} M_2 \Longleftrightarrow \widetilde{M}_1 \preceq^{b}_{\Sigma} \widetilde{M}_2$.

Proposition 8 may have a considerable utility in checking qb-simulation, since it provides a way of taking an advantage of efficient simulation checking algorithms (see Cleaveland and Sokolsky (2001) and Etessami et al. (2001)) that are applicable to block simulation. Quasi-block (unlike visible or block simulations) is preserved under asynchronous compositions of LTSs.

Proposition 9. *Let $M_i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i\rangle, i = 1, 2, 3, 4$, be four LTSs such that $(\Sigma^1 \cup \Sigma^2) \cap (\Sigma^3 \cup \Sigma^4) = \emptyset$, $A^1 = A^2 = A'$, $A^3 = A^4 = A''$, and $A' \cap A'' = \emptyset$.*
Let Σ' and Σ'' be distinguished sets such that $\Sigma' \subseteq (\Sigma^1 \cup \Sigma^2)$ and $\Sigma'' \subseteq (\Sigma^3 \cup \Sigma^4)$. Let $\Gamma = (\Delta, {}^{-})$ be a synchronizer such that $\Delta \subseteq A'$, and ${}^{-} : \Delta \to A''$. Then $M_1 \preceq^{qb}_{\Sigma'} M_2$ and $M_3 \preceq^{qb}_{\Sigma''} M_4$ imply $M_1 \parallel_{\Gamma} M_3 \preceq^{qb}_{\Sigma' \cup \Sigma''} M_2 \parallel_{\Gamma} M_4$.

Proof. Let H' be a relation of qb-simulation on M_1, M_2 w.r.t. Σ', E^1, E^2, and H'' be a relation of qb-simulation on M_3, M_4 w.r.t Σ'', E^3, E^4. We consider a relation $H \subseteq (S^1 \times S^3) \times (S^2 \times S^4)$ such that

$$H = \{((s^1, s^3), (s^2, s^4)) \mid (s^1, s^2) \in H' \wedge (s^3, s^4) \in H''\}$$

and demonstrate that H is a relation of qb-simulation on $M_1 \parallel_{\Gamma} M_3$ and $M_2 \parallel_{\Gamma} M_4$ w.r.t. the appropriate sets of events E' and E''.

The set of transitions E' in the model $M_1 \parallel_{\Gamma} M_3$ is defined as follows. Every transition $((s^1, s^3), a, (t^1, t^3))$ is included in E' iff it meets one of the conditions below:

- $a \in A' \wedge (s^1, a, t^1) \in E^1$,
- $a \in A'' \wedge (s^3, a, t^3) \in E^3$,
- $a = \tau$ and for some $b \in A'$ it is true that $(s^1, b, t^1) \in R^1 \wedge (s^3, \overline{b}, t^3) \in R^3$.

The set of transitions E'' in the model $M_2 \parallel_{\Gamma} M_4$ is defined analogously.

The key idea of the proof is as follows. A qb-simulation for the compositions $M_1 \parallel_{\Gamma} M_3$ and $M_2 \parallel_{\Gamma} M_4$ may fail only in the case when some finite block outgoing from a state (s^1, s^3) in the model $M_1 \parallel_{\Gamma} M_3$ is too much extensive and does not fit to any block outgoing from the corresponding state (s^2, s^4) in the model $M_2 \parallel_{\Gamma} M_4$. We show that the sets of events E' and E'' are chosen such that all "lengthy" blocks are "cut" into small pieces and these pieces satisfy the requirements QB1, QB2, and QB3 from the Definition 5 for any pair of corresponding states $((s^1, s^3), (s^2, s^4)) \in H$.

1. It is easy to see that the relation H satisfies the requirement QB1 due to the following chain of equalities:

$$\begin{aligned} L^{13}((s^1, s^3)) \cap (\Sigma' \cup \Sigma'') &= (L^1(s^1) \cup L^3(s^3)) \cap (\Sigma' \cup \Sigma'') \\ &= (L^1(s^1) \cap \Sigma') \cup (L^3(s^3) \cap \Sigma'') \\ &= \{\text{since } (s^1, s^2) \in H' \text{ and } (s^3, s^4) \in H''\} \\ (L^2(s^2) \cap \Sigma') \cup (L^4(s^4) \cap \Sigma'') &= (L^2(s^2) \cup L^4(s^4)) \cap (\Sigma' \cup \Sigma'') \\ &= L^{24}((s^2, s^4)) \cap (\Sigma' \cup \Sigma''), \end{aligned}$$

where L^{13} and L^{24} are the evaluation functions of LTSs $M_1 \parallel_{\Gamma} M_3$ and $M_2 \parallel_{\Gamma} M_4$ respectively.

2. To make sure that H satisfies the requirement QB2 consider an arbitrary finite block B_{13} from the set $MAXF(E', (s^1, s^3))$:

$$B_{13} = (s^1_1, s^3_1) \xrightarrow{\tau} (s^1_2, s^3_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^1_m, s^3_m) \xrightarrow{a} (s^1_{m+1}, s^3_{m+1}),$$

where $(s^1_1, s^3_1) = (s^1, s^3)$, and its projections $pr_{M_1}(B_{13})$ and $pr_{M_3}(B_{13})$ on LTSs M_1 and M_2 respectively. Let B_1 be an arbitrary block in M_1 such that $pr_{M_1}(B_{13})$ is a prefix of B_1, and B_3 be an arbitrary block in M_3 such that $pr_{M_3}(B_{13})$ is a prefix of B_3. Then several cases are possible depending on the structure of B_1 and B_3.

(1) $B_1 = pr_{M_1}(B_{13})$ and $B_3 = pr_{M_3}(B_{13})$. This case is possible only when $a = \tau$ and transition $(s^1_m, s^3_m) \xrightarrow{\tau} (s^1_{m+1}, s^3_{m+1})$ is a result of synchronous communication of M_1 and M_3 via some transitions $s^1_m \xrightarrow{b} s^1_{m+1}$ and $s^3_m \xrightarrow{\bar{b}} s^3_{m+1}$. Since $(s^1, s^2) \in H'$, $(s^3, s^4) \in H''$, there exists a pair of blocks B_2 and B_4 in $MAXF(E^2, s^2)$ and $MAXF(E^4, s^4)$ respectively such that B_2 matches B_1 w.r.t. H' and B_4 matches B_3 w.r.t. H''. By Definition 5 this means that B_2 ends with a transition $s^2_k \xrightarrow{b} s^2_{k+1}$, whereas B_4 ends with a transition $s^4_\ell \xrightarrow{\bar{b}} s^4_{\ell+1}$. Then by the definition of asynchronous composition the model $M_2 \parallel_\Gamma M_4$ has a block B_{24} such that
   - B_{24} ends with the transition $(s^2_k, s^4_\ell) \xrightarrow{\tau} (s^2_{k+1}, s^4_{\ell+1})$ which is a result of synchronous communication of M_2 and M_4 via $s^2_k \xrightarrow{b} s^2_{k+1}$ and $s^4_\ell \xrightarrow{\bar{b}} s^4_{\ell+1}$,
   - $B_2 = pr_{M_2}(B_{24})$ and $B_4 = pr_{M_4}(B_{24})$.

   In fact, B_{24} can be obtained by interleaving arbitrarily the transitions from B_2 with that from B_4. It can be readily seen that B_{24} matches B_{13} w.r.t. H.

(2) $pr_{M_3}(B_{13}) \neq B_3$. This implies that B_1 is a finite block. Since $(s^1, s^2) \in H'$ and $(s^3, s^4) \in H''$, there exists a pair of blocks

$$B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{a} s^2_{k+1},$$
$$B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_\ell \xrightarrow{\tau} \cdots,$$

   such that B_2 is a finite block from $MAXF(E^2, s^2)$ and it matches B_1 w.r.t. H', whereas B_4 is either a finite block from $MAXF(E^4, s^4)$ (if the block B_3 is finite) or an infinite block from $MAXI(E^4, s^4)$ (if the block B_3 is infinite), and it matches B_3 w.r.t. H''. Then by the definition of asynchronous composition the model $M_2 \parallel_\Gamma M_4$ has a finite block

$$B_{24} = (s^2_1, s^4_1) \xrightarrow{\tau} (s^2_1, s^4_2) \xrightarrow{\tau} (s^2_2, s^4_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^2_k, s^4_2) \xrightarrow{a} (s^2_{k+1}, s^4_2),$$

   and this block matches B_{13} w.r.t. H.

(3) $pr_{M_1}(B_{13}) \neq B_1$. Similar reasoning shows that in this case LTS $M_2 \parallel_\Gamma M_4$ also has a finite block B_{24} which matches B_{13} w.r.t. H.

3. Finally, we check that H satisfies the requirement QB3. Consider an arbitrary infinite block B_{13} from the set $MAXI(E', (s^1, s^3))$:

$$B_{13} = (s^1_1, s^3_1) \xrightarrow{\tau} (s^1_2, s^3_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^1_m, s^3_m) \xrightarrow{\tau} \cdots,$$

where $(s^1_1, s^3_1) = (s^1, s^3)$, and its projections $pr_{M_1}(B_{13})$ and $pr_{M_3}(B_{13})$ on LTSs M_1 and M_2 respectively. Let B_1 be an arbitrary block in M_1 whose prefix is $pr_{M_1}(B_{13})$, and B_3 be an arbitrary block in M_3 whose prefix is $pr_{M_3}(B_{13})$. Then several cases are possible depending on whether B_1 and B_3 are finite or infinite blocks.

(1) Both blocks B_1 and B_3 are infinite. Since $(s^1, s^2) \in H'$ and $(s^3, s^4) \in H''$, there exists a pair of infinite blocks B_2 and B_4 in $MAXI(E^2, s^2)$ and $MAXI(E^4, s^4)$ respectively such that B_2 matches B_1 w.r.t. H' and B_4 matches B_3 w.r.t. H''. Then by the definition of asynchronous composition the model $M_2 \parallel_\Gamma M_4$ has a block B_{24} which is obtained from B_2 and B_4 by interleaving arbitrarily the transitions from B_2 with that from B_4. Clearly, B_{24} matches B_{13} w.r.t. H.

(2) One of the blocks (say, B_1) is infinite, whereas the other is finite. Since $(s^1, s^2) \in H'$ and $(s^3, s^4) \in H'$, there exists a pair of blocks

$$B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{\tau} \cdots,$$
$$B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_\ell \xrightarrow{b} s^4_{\ell+1},$$

such that B_2 is an infinite block from $MAXI(E^2, s^2)$ matching B_1 w.r.t. H', whereas B_4 is a finite block from $MAXF(E^4, s^4)$ matching B_3 w.r.t. H''. Then by the definition of asynchronous composition the model $M_2 \parallel_\Gamma M_4$ has an infinite block

$$B_{24} = (s^2_1, s^4_1) \xrightarrow{\tau} (s^2_1, s^4_2) \xrightarrow{\tau} (s^2_2, s^4_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^2_k, s^4_2) \xrightarrow{\tau} (s^2_{k+1}, s^4_2), \ldots$$

which matches B_{13} w.r.t. H.

Thus, we demonstrated that the relation H satisfies all the requirements of quasi-block simulation. Note, that for every pair of initial states $s^1_0 \in S^1_0$, $s^3_0 \in S^3_0$ there exists a pair of initial states $s^2_0 \in S^2_0$, $s^4_0 \in S^4_0$ such that $(s^1_0, s^2_0) \in H'$ and $(s^3_0, s^4_0) \in H''$. This implies $M_1 \parallel_\Gamma M_3 \preceq^{qb}_{\Sigma' \cup \Sigma''} M_2 \parallel_\Gamma M_4$. □

Yet another simulation which has close relationships with quasi-block one is stuttering simulation. It was introduced by Penczek et al. (1999) on the basis of stuttering bisimulation which enjoys wide applications in the framework of partial order reduction technique (see Browne et al. (1988) and Clarke et al. (1999)).

Let $M_i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i \rangle$, $i = 1, 2$, be two LTSs, and $\Sigma \subseteq \Sigma^1 \cap \Sigma^2$. A relation $H \subseteq S^1 \times S^2$ is called a *stuttering simulation* w.r.t. Σ iff every pair $(s', s'') \in H$ complies with the following requirements:

(1) $L^1(s') \cap \Sigma = L^2(s'') \cap \Sigma$.

(2) For every path $\pi' = s'_1 \xrightarrow{a_1} s'_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} s'_k \xrightarrow{a_k} \cdots$, $s'_0 = s'$ there is a path $\pi'' = s''_1 \xrightarrow{a_1} s''_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} s''_k \xrightarrow{a_k} \cdots$, $s''_0 = s''$ and partitions $P'_1 P'_2 \ldots, P''_1 P''_2 \ldots$ of π' and π'', 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 $M_1 \preceq^{st}_\Sigma M_2$ to indicate the existence of stuttering simulation between M_1 and M_2.

Proposition 10. $M_1 \preceq^{qb}_\Sigma M_2 \Longrightarrow M_1 \preceq^{st}_\Sigma M_2$.

A proof of this proposition is straightforward. Since stuttering simulation preserves ACTL*-X, this leads us to the following conclusion.

Proposition 11. *Suppose that a qb-simulation $M_1 \preceq^{qb}_{\Sigma_\varphi} M_2$ is well-formed w.r.t. a ACTL*-X formula φ, and $M_2 \models \varphi$. Then $M_1 \models \varphi$ as well.*

As 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.

Example 12. Consider two models M_1 and M_2 depicted in Fig. 3.

The evaluation functions associate with every state exactly one of the basic propositions from the set $\Sigma_0 = \{p_1, p_2, p_3\}$. It could be directly checked that $M_1 \preceq^{st}_{\Sigma_0} M_2$.

Suppose that there exists a qb-simulation on M_1 and M_2 w.r.t. some sets of events E_1 and E_2. It should be noted, that the path $\pi = s_1 s_6 s_7$ may correspond only to the path $\pi' = u_1 u_4 u_6 u_7$ with the partitioning s_1; s_6; s_7 and u_1; $u_4 u_6$; u_7 respectively. This means that the transition $u_4 \xrightarrow{\tau} u_6$ is not in E_2. On the other hand, the path $\delta = s_1 s_2 s_4 s_6 s_7$ may correspond only to the path $\delta' = u_1 u_2 u_4 u_6 u_7$ with the partitioning s_1; $s_2 s_4$; s_6; s_7 and u_1; $u_2 u_4$; u_6; u_7 respectively. Hence, $u_4 \xrightarrow{\tau} u_6$ has to be in E_2. This certifies that $M_1 \not\preceq^{qb}_{\Sigma_0} M_2$. □

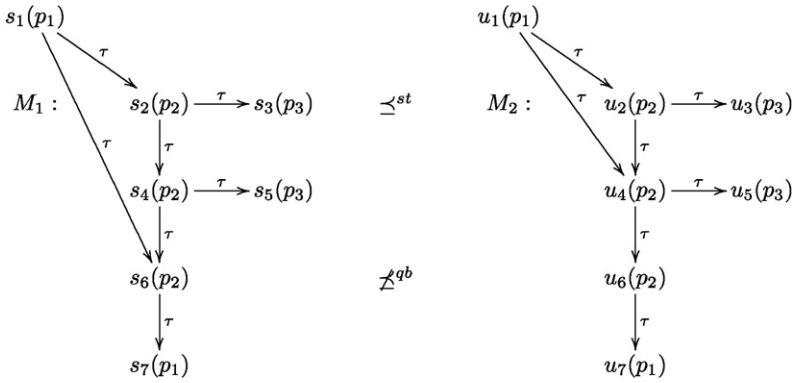


Fig. 3. qb-simulation versus st-simulation.

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.

As a result we may outline the following invariant-based approach to the verification of parameterized asynchronous networks with the help of qb-simulation. Given an infinite family $\mathcal{F} = \{P_i\}_{i=1}^{\infty}$ of network models, where each model P_i is an asynchronous parallel composition of the form $p_1 \| p_2 \| \ldots \| p_i \| q$, find the least k such that $P_{k+1} \prec^{b} P_k$. Since $P_{k+1} \prec^{b} P_k$ implies $P_{k+1} \prec^{qb} P_k$, Proposition 9 guarantees that $P_n \prec^{qb} P_k$ holds for every n, $n > k$. If a property to be checked is expressed by a formula φ from ACTL*-X then, in view of Proposition 11, to verify this property of the parameterized system $\mathcal{F}$ it is sufficient to model check a finite set of finite models P_i, $1 \le i \le k$, against φ. As it is easy to see, the most difficult step is to establish a block simulation between finite models M_k and M_{k+1}. In Section 4 we show how to implement this approach in practice.

4. Applying quasi-block simulation to the verification of asynchronous networks

There are very few papers (in fact, Calder and Miller (2002) were not aware of any) where the induction-based verification technique is applied to asynchronous networks. Emerson and Namjoshi (1995) thoroughly studied 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 ACTL*-X properties a *cut-off* effect takes place, i.e. the verification of the whole parameterized system can be reduced to the model checking of finitely many instances of the system. In a certain sense, a cut-off plays a role of an invariant for such systems. Clarke et al. (2004) extended the results of Emerson and Namjoshi from rings to some other classes of asynchronous networks. Nevertheless, many interesting classes of parameterized asynchronous systems do not fall into this category.

The application of invariant-based techniques to the verification of asynchronous parameterized systems was studied also by Shtadler and Grumberg (1990), Marelly and Grumberg (1991), and Clarke et al. (1995, 1997). But in all these cases invariants were generated only with the help of specification driven abstraction. Thus, for example, Clarke et al. (1997) represented parameterized systems by network grammars, and used regular languages to express state properties. To generate invariants they developed an *unfolding heuristic*: 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 an appropriate abstraction. Much attention has been paid to the development of effective technique for constructing required abstractions.

4.1. Finding invariants in binary trees

To demonstrate the feasibility of quasi-block simulation we limit our scope to the parameterized systems whose communication networks are binary trees. The nodes of such networks are the

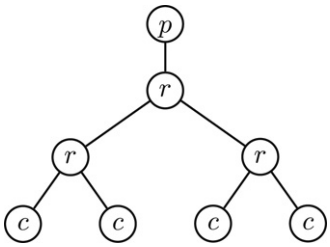


Fig. 4. The network of model $p(r(r(c, c), r(c, c)))$.

processes of three types:

(1) the only producer attached to the root of the tree,
(2) intermediate nodes (routers), and
(3) leaf nodes (consumers).

Specifically we consider parameterized systems generated by the following network grammar G:

$$
\begin{aligned}
&P \to p \parallel_1 T\\
&T \to T \parallel_2 r \parallel_3 T\\
&T \to c \parallel_2 r \parallel_3 c\\
&T \to c
\end{aligned}
$$

where terminals p (producer), c (consumer), and r (router) are arbitrary finite state processes; symbols $\parallel_1$, $\parallel_2$, and $\parallel_3$ are shorthand for parallel composition with respect to action synchronizers and action renaming functions specific to each symbol. In the grammar G the producer has the only child process and each router has exactly two child processes. We denote by $\mathcal{L}(G)$ 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 on the right-hand 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 depth 3 depicted in Fig. 4. Sometimes a behavior of a producer is of no importance for the analysis of a model. In this case 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 $r(t_1, t_2) \preceq^{qb}_{\Sigma(r)} r(t'_1, t'_2)$ 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.

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

Denote by T_N the set of tree networks of depth N derived from T.

Proposition 13. *If the following qb-simulations are valid*

$$r(r(c, c), c) \preceq^{qb}_{\Sigma(r)} r(c, r(c, c)) \tag{1}$$

$$r(r(c, c), r(c, c)) \preceq^{qb}_{\Sigma(r)} r(c, r(c, c)) \tag{2}$$

$$r(c, r(c, r(c, c))) \preceq^{qb}_{\Sigma(r)} r(r(c, c), r(c, c)) \tag{3}$$

$$r(r(c, r(c, c)), c) \preceq^{qb}_{\Sigma(r)} r(r(c, c), r(c, c)) \tag{4}$$

$$r(r(c, r(c, c)), r(c, r(c, c))) \preceq^{qb}_{\Sigma(r)} r(r(c, c), r(c, c)) \tag{5}$$

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

Proof. Assertion (1) allows us to rotate a tree. Applying assertions (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 $t' \in \{r(c, r(c, r(c, c))), r(r(c, r(c, c)), c), r(r(c, r(c, c)), r(c, r(c, c)))\}$ that $t \preceq^{qb}_{\Sigma(r)} t'$. In view of the assertions (3)–(5) the latter is reduced to qb-simulation $t \preceq^{qb}_{\Sigma(r)} r(r(c, c), r(c, c))$.

All these reductions can be made due to monotonicity of parallel composition with respect to qb-simulation (Proposition 9). □

As it may be seen from Proposition 13 the model $Inv = r(r(c, c), r(c, c))$ is a keystone of our construction.

Proposition 14. *If the assertions (1)–(5) are valid, then the model Inv qb-simulates every network t from T_N, $N \geq 3$, i.e. $t \preceq^{qb}_{\Sigma(r)} Inv$.*

Proof. The proof proceeds by induction on the depth N of a tree network t. The basis of induction ($N = 3$) is trivial due to Proposition 13.

Suppose that Inv qb-simulates every network from T_N, and 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 bottom subtree of depth 3 in t. The network t' is in T_N, so, by the induction hypothesis, $t' \preceq^{qb} Inv$. By Proposition 13, every subtree of depth 3 is qb-simulated by Inv. Therefore, we have $t \preceq^{qb} Inv$, due to monotonicity of parallel composition with respect to qb-simulation. □

By combining Propositions 9 and 14, we arrive at the following conclusion.

Proposition 15. *If the assumptions (1) and (5) are valid then the model $p(Inv)$ is an invariant of the family $\mathcal{L}(G)$, i.e. $p(t) \preceq^{qb}_{\Sigma(r)\cup\Sigma(p)} p(Inv)$ holds for every network t of depth N, $N \geq 3$, derived from non-terminal T.*

Thus, to certify that the finite model $p(Inv)$ is an invariant of the infinite family of models $\mathcal{L}(G)$ it is sufficient to make certain that the assumptions (1)–(5) are valid. In this case Propositions 11 and 15 guarantee that any property of the producer and the top router is satisfied on any tree network of depth greater than 2 iff the property is satisfied on the invariant.

Proposition 16. *Let φ be an arbitrary ACTL*-X formula which includes only atomic propositions from $\Sigma(p) \cup \Sigma(r)$. If the assertions (1)–(5) are valid then*

$$p(Inv) \models \varphi \iff p(w) \models \varphi \text{ holds for every network } w, \quad w \in \bigcup_{i=2}^{\infty} T_i.$$

4.2. Checking quasi-block simulation

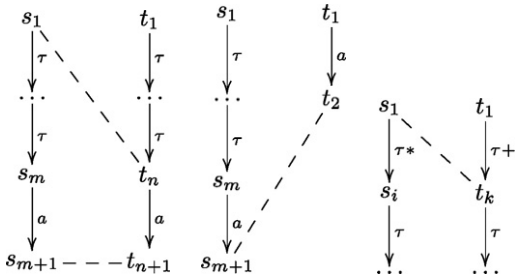
When checking qb-simulation between finite models we rely on Propositions 7 and 8. Since $M_1 \preceq^{b}_{\Sigma} M_2$ implies $M_1 \preceq^{qb}_{\Sigma} M_2$, qb-simulation checking of the assumptions (1)–(5) can be reduced to block simulation checking of the corresponding models.

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 $M_i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i \rangle$, $i = 1, 2$, be two LTSs. Let Σ 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 17. 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. Σ, $E^1 = Observ(M_1, \Sigma)$, $E^2 = Observ(M_2, \Sigma)$, iff every pair $(s_1, t_1) \in H$ meets the following requirements:

(1) $L^1(s_1) \cap \Sigma = L^2(t_1) \cap \Sigma$,

(2) for every finite block $B' = s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1}$ from $MAXF(E^1, s^1)$ there is a block $B'' = t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{a} t_{n+1}$ in $MAXF(E^2, t_1)$ such that:



(a) Item 2.a (b) Item 2.b (c) Item 2.c

Fig. 5. Semi-block simulation.

(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) (divergency) for every infinite block $B' = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_i \xrightarrow{\tau} \cdots$ from $MAXI(E^1, s_1,)$ there exists an infinite block $B'' = t_1 \xrightarrow{\tau} t_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_j \xrightarrow{\tau} \cdots$ in $MAXI(E^2, t_1,)$ and a number $k > 1$ such that $(s_1, t_k) \in H$.

The items of this definition are illustrated in Fig. 5. Pairs of states connected by dashed lines must belong to the relation H. Note, that in the definition of block simulation virtually all the combinations of pairs in the matching blocks are required to be in the relation H. The definition of semi-block simulation imposes weaker constraints on matching blocks, i.e. it is sufficient to check that at most two pairs of states in matching blocks are in the relation.

We write $M_1 \preceq^{sb}_{\Sigma} M_2$ iff there exists a binary relation $H \subseteq S^1 \times S^2$ of LTSs M_1 and M_2 such that H is a semi-block simulation on M_1 and M_2 w.r.t. Σ, $Observ(M_1, \Sigma)$, $Observ(M_2, \Sigma)$, and for every initial state $s_0 \in S^1_0$ there exists an initial state $t_0 \in S^2_0$ such that $(s_0, t_0) \in H$.

Though the constraints on matching blocks in definitions of block simulation and semi-block simulation differ significantly, the former relation may be constructed on the basis of the latter relation and vice versa.

Proposition 18. $M_1 \preceq^{sb}_{\Sigma} M_2 \Longleftrightarrow M_1 \preceq^{b}_{\Sigma} M_2$.

5. Example: Resource reservation protocol

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).

Resource reservation protocol (Braden et al., 1997) is designed to reserve resources in a network with a guaranteed quality of service. 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-oriented 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 bidirectional 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 router processes change paths to adapt 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 eight 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 acknowledgements. A *path_error* message is sent upstream whenever an error occurs during path allocation. A *resv_error* message is sent downstream when 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 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 the sessions periodically by exchanging *path_refr* and *resv_refr* messages.

5.1. A model of RSVP

Several properties of RSVP have already been verified in Creese and Reed (1999) and Villapol (2003). Creese and Reed (1999) used CSP to construct a formal model of some fragment of RSVP to check the property of reservation merging in a binary multicast tree. Villapol (2003) studied a model of RSVP with one producer, one router and one consumer 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 Creese and Reed (1999) 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 versus unicast. When dealing with a unicast messaging, one has to attach a destination (and possibly source) address to every message. In this case one should either consider models with potentially infinite state space (as address size grows with the number of processes), or 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 resources available 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 and timeouts.

Table 1
Validating the assertions (1)–(5). Results.

Assumption on M_1 and M_2	#States of M_1	#States of M_2	#Positive pairs	#Negative pairs	Time	Memory (Mb)
(1)	1732	1277	1557	40	0.03 s	15
(2)	24993	1277	28127	471	2.8 s	16
(3)	14672	24993	27252	2281	4.68 s	18
(4)	21659	24993	34274	1947	6 s	21
(5)	1.8×10^6	24993	6.8×10^6	224761	11.7 h	296

The models of RSVP were implemented in the model description language PROMELA in the framework of the open-source software verification tool SPIN (cf. Holzmann (2003)). The usage of PROMELA gave us an advantage of exploiting a simulator and model checker from SPIN.

In order to verify parameterized RSVP model according to the technique described in Section 4 we break the checking procedure into two stages. At the first stage we build semi-block simulations between the models mentioned in Proposition 13. If all the simulations are built successfully and assertions (1)–(5) are valid, then the model $p(Inv)$ is proved to be an invariant and the second stage is applied. At the second stage SPIN is used to check the invariant against given specifications.

The implementation of our experimental system for induction-based verification of parameterized models CHEAPS (Checker of Asynchronous Parameterized Systems) as well as a description of parameterized model of RSVP and the results of checking this model are available at http://lvk.cs.msu.su/~konnov/cheaps/.

5.2. Stage 1: Computing semi-block simulations to find invariants

We implemented a naive algorithm for checking semi-block simulation which works iteratively according to the definition above. It adds all the 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 $|S_1| \cdot |S_2|$. 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 the 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 GB). To bypass the memory limitation we implemented a relation storage as a minimized automaton from SPIN (see Puri and Holzmann (1998)). We found that this not only drastically decreases memory consumption, but also speeds up the computation.

The results of our experiments are depicted in Table 1. All assumptions were proved to be valid by finding appropriate semi-block simulations. The program was running on 2.4 GHz AMD Opteron provided by Laboratory of Computer Systems, Moscow State University.

The checking of $r(r(c, r(c, c), r(c, r(c, c))))$ versus Inv takes a considerable amount of time as model state space grows rapidly with a number of processes. We have successfully optimized the computation by using various optimization techniques: splitting positives into stable and unstable subsets, using back propagation of negative results, pair enumeration using sequential file representation, caching of previous queries to DFA. Thanks to the optimizations we reduced time of the computation to reasonable values, though there is still room for further improvement. We believe, it is possible to exploit symbolic and bisimulation-based techniques to find equivalent states in models, and thus reduce state space of models significantly.

5.3. Stage 2: Using invariants for checking properties of RSVP.

As it follows from Proposition 11 and Corollary 15, to check the whole infinite family of networks $\mathcal{L}(G)$ against any ACTL*-X -specification φ 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 depth is less than 3, satisfy φ.

Our main efforts were focused on the computation of block simulation and thus finding a desired invariant. After obtaining a desirable invariant we checked it against a number of specifications (safety requirements). 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 tear acknowledge* while it does not send *path teardown* message. This specification is expressed in Linear Temporal Logic (and in ACTL*-X) as follows:

$$\varphi_1 = \mathsf{A}(\mathsf{G}\neg \mathit{producer}!\mathit{path\_tear} \rightarrow \mathsf{G}\neg \mathit{producer}?\mathit{path\_tear\_acknowledge}). \tag{6}$$

The second checked property is that of reservation merging, which had been checked by Creese and Reed (1999). In terms of our model it means that reservation request cannot appear while the router is in "reserved" state:

$$\varphi_2 = \mathsf{AG}(\mathit{router}_1.\mathit{reserved} \rightarrow \neg \mathit{router}_1.\mathit{parent}!\mathit{resv}). \tag{7}$$

The third and the fourth checked properties are reformulated versions of those that were studied by Villapol (2003). The third property requires that a router may receive a reservation request only when a path is set up:

$$\varphi_3 = \mathsf{AG}(\mathit{router}_1?\mathit{resv} \rightarrow \mathit{router}_1.\mathit{state} \neq \mathit{ROUTER\_FREE}). \tag{8}$$

The fourth property states: if data is transmitted via a router, then a reservation is already set up:

$$\varphi_4 = \mathsf{AG}(\mathit{router}_1?\mathit{data} \rightarrow \mathit{router}_1.\mathit{state} = \mathit{ROUTER\_RESV}). \tag{9}$$

By applying Spin model checker we found that specification φ_1 is satisfied on the system $p(r(r(c,c),r(c,c)))$ and $\varphi_2, \varphi_3, \varphi_4$ are satisfied on the system $r(r(c,c),r(c,c))$. Due to Proposition 16 we conclude that properties $\{\varphi_i\}_{i=1}^4$ are satisfied on any tree of depth greater than 2.

6. Conclusions and directions for future research

In this article we have introduced a new sort of simulation relation for Labelled Transition Systems (LTSs) — a quasi-block simulation (qb-simulation). We have proved that qb-simulation satisfies two principal requirements: (1) it preserves the satisfiability of ACTL*-X formulae, and (2) asynchronous parallel composition of LTSs is monotonic w.r.t. qb-simulation. Owing to these properties of qb-simulation, the induction-based approach to parameterized model checking can be extended to infinite families of *asynchronous* communicating processes. We have implemented a qb-simulation checking algorithm and used it to find an invariant of a parameterized model of Resource ReSerVation Protocol (RSVP). This case study favours the view that induction-based technique supplied with an efficient qb-simulation checking procedure is attractive for verification of complex families of LTSs.

Nevertheless, there is a number of tasks to be solved next to build qb-simulation up.

(1) The complexity of qb-simulation checking problem needs further consideration; as it can be seen from Definition 5, this problem is in NP. If it happens to be NP-complete then we need a suitable approximation to qb-simulation that can be checked in polynomial time.
(2) Network grammars provide a suitable means for the description of families of tree-like networks composed of heterogeneous processes. In Section 4 we have demonstrated that in the case of simple grammar G an invariant of the family $\mathcal{L}(G)$ can be computed pure automatically without resorting to abstractions. It would be interesting to extend this approach to arbitrary parameterized networks specified by context-free network grammars.
(3) We are interested also in finding such a parameterized asynchronous system whose invariants cannot be computed with the help of qb-simulation. This system will serve as a counterexample to reveal the limitations of qb-simulations and to outline the ways for its further improvement.

Acknowledgements

We thank the anonymous reviewers for their helpful comments that have resulted in significant improvements of the paper.

References

Apt, K., Kozen, D., 1986. Limits for automatic verification of finite-state concurrent systems. Inform. Process. Lett. 15, 307–309.
Apt, K., Olderog, E.R., 1997. Verification of Sequential and Concurrent Programs, second edition. In: Graduate Texts in Computer Science, Springer-Verlag, 364 pages.
Bosnacki, D., Dams, D., Holenderski, L., 2001. A heuristic for symmetry reductions with scalarset. In: Proceedings of FME2001. In: Lecture Notes in Computer Science, vol. 2021. pp. 518–533.
Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S., 1997. Resource reservation protocol (RSVP). http://tools.ietf.org/html/rfc2205.
Browne, M.C., Clarke, E.M., Grümberg, O., 1988. Characterizing finite kripke structures in propositional temporal logic. Theoret. Comput. Sci. 59 (1–2), 115–131.
Calder, M., Miller, A., 2002. Five ways to use induction and symmetry in the verification of networks of processes by model checking. In: Automated Verification of Critical Systems, AvoCS 2002, pp. 29–42.
Chehaibar, G., Garavel, H., Mounier, L., Tawbi, N., Zulian, F., 1996. Specification and verification of the powerscale bus arbitration protocol: An industrial experiment with LOTOS. In: Proceedings of FORTE/PSTV'96. Kaiserslautern, Germany. Chapman & Hall, London.
Cimatti, A., 2000. Industrial applications of model checking. In: Proceedings of the 4-th Summer School on Modeling and Verification of Parallel Processes. In: Lecture Notes in Computer Science, vol. 2067. pp. 153–168.
Clarke, E.M., Grumberg, O., Long, D., 1992. Model checking and abstraction. In: Proceedings of Principles of Programming Languages, pp. 343–354.
Clarke, E.M., Grumberg, O., Hirashi, H., Jha, S., Long, D., McMillan, D.E., Ness, L.A., 1993. Verification of the Futurebus+ cache coherence protocol. In: Proceedings of the IFIP Conference on Hardware Description Languages and their Applications, Ottawa, Canada, 26–28 April, 1993.
Clarke, E.M., Filkorn, T.S.J., 1993. Exploiting symmetry in temporal logic model checking. In: Proceedings of CAV'93. In: Lecture Notes in Computer Science, vol. 697. pp. 450–461.
Clarke, E.M., Grumberg, O., Jha, S., 1995. Verifying parameterized networks using abstraction and regular languages. In: Proceedings of 6-th International Conference on Concurrency Theory, pp. 395–407.
Clarke, E.M., Grumberg, O., Jha, S., 1997. Verifying parameterized networks. ACM Trans. Program. Lang. Syst. 19 (5), 726–750.
Clarke, E.M., Grumberg, J., Peled, D.A., 1999. Model Checking. MIT Press, Cambridge, MA, USA.
Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H., 2000. Counterexample — Guided abstraction refinement. In: Proceedings of CAV'00. In: Lecture Notes in Computer Science, vol. 1855. pp. 154–169.
Clarke, E.M., Talupur, M., Touili, T., Veith, H., 2004. Verification by network decomposition. In: Proceedings of CONCUR 2004. In: Lecture Notes in Computer Science, vol. 3170. pp. 276–291.
Cleaveland, R., Sokolsky, O., 2001. Equivalence and preorder checking for finite-state systems. In: Bergstra, J., Ponse, A., Smolka, S. (Eds.), Handbook of Process Algebra. North-Holland, pp. 391–424.
Creese, S., Reed, J., 1999. Verifying end-to-end protocols using induction with csp/fdr. In: Proceedings of IPPS/SPDP Workshop, pp. 1243–1257.
Dams, D., Grumberg, O., Gerth, R., 1994. Abstract interpretation of reactive systems: Abstractions preserving *ACTL**, *ECTL** and *CTL**. In: Proceedings of IFIP Working Conference and Programming Concepts, Methods and Calculii.
Donaldson, A.F., Miller, A., 2005. Automatic symmetry detection for model checking using computational group theory. In: Proceedings of International Symposium of Formal Methods. In: Lecture Notes in Computer Science, vol. 3582. pp. 481–496.
Emerson, E., Namjoshi, K., 1995. Reasoning about rings. In: Proceedings of 22th ACM Conf. on Principles of Programming Languages, pp. 85–94.
Emerson, E., Sistla, A., 1996. Symmetry and model checking. Form. Methods Syst. Des. 9 (1–2), 105–131.
Emerson, E.A., Kahlon, V., 2003. Exact and efficient verification of parameterized cache coherence protocols. L'Aquila, Italy.
Emerson, E. A., Kahlon, V., 2004. Parameterized model checking of ring-based message passing systems. In: Lecture Notes in Computer Science, vol. 3210. pp. 325–339.
Etessami, K., Schuller, R., Wilke, Th., 2001. Fair simulation relations, parity games, and state space reduction for buchi automata. In: Proceedings of ICALP'01. In: Lecture Notes in Computer Science, vol. 2076. pp. 694–707.
German, S.M., Sistla, A.P., 1992. Reasoning about systems with many processes. J. ACM 39 (3), 675–735.
Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G., 2003. Software verification with BLAST. In: Proceedings of the 10th SPIN Workshop on Model Checking Software. SPIN. In: Lecture Notes in Computer Science, vol. 2648. pp. 235–239.
Holzmann, G.J., 2003. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional.
Ip, C.N., Dill, D.L., 1997. Verifying systems with replicated components in murφ. Form. Methods Syst. Des. 14 (3), 273–310.
Kaner, C., Falk, J., Nguyen, H.Q., 1999. Testing Computer Software, 2nd edition. John Wiley and Sons, Inc., New York, 480 pages.
Kesten, Y., Pnueli, A., 2000. Verification by finitary abstraction. Inform. Comput. 163, 203–243.
Konnov, I., Zakharov, V., 2005. An approach to the verification of symmetric parameterized distributed systems. Program. Comput. Softw. 31 (5), 3–17.
Kurshan, R.P., McMillan, K., 1989. A structural induction theorem for processes. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 239–247.
Lesens, D., Saidi, H., 1997. Automatic verification of parameterized networks of processes by abstraction. In: Proceedings of 2nd International Workshop on the Verification of Infinite State Systems, INFINITY'97, pp. 268–278.
Manku, G.S., Hojati, R., Brayton, R.K., 1998. Structural symmetry and model checking. In: Proceedings of CAV'98, pp. 159–171.

Marelly, R., Grumberg, O., 1991. Gormel — grammar oriented model checker. Tech. Rep. 697, The Technion, Haifa, Israel.
Pardo, A., 1997. Automatic abstraction techniques for propositional — calculus model checking. In: Proceedings of CAV'98. In: Lecture Notes in Computer Science, vol. 1254. pp. 12–23.
Penczek, W., Szreter, M., Gerth, R., Kuiper, R., 1999. Partial order reductions preserving simulations.
Puri, A., Holzmann, G., 1998. A minimized automaton representation of reachable states. Softw. Tools Technol. Transfer 3 (1).
Shtadler, Z., Grumberg, O., 1990. Network grammars, communication behaviors and automatic verification. In: Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems, pp. 151–165.
Villapol, M., 2003. Modelling and analysis of the resource reservation protocol using coloured petri nets. Ph.D. Thesis, Institute for Telecommunications Research and Computer Systems Engineering Centre, University of South Australia.
Wolper, P., Lovinfosse, V., 1989. Verifying properties of large sets of processes with network invariants. In: Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems. In: Lecture Notes in Computer Science, vol. 407. pp. 68–80.