On the verification of asynchronous parameterized networks of communicating processes by model checking[1]

I. V. Konnov , V. A. Zakharov

Abstract. The 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. We consider some aspects of the induction-based technique which assumes the construction of finite invariants of such systems. An invariant process is one which 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 apply it to verification of Dijkstra's token ring algorithm.

Keywords: program verification, asynchronous networks, model checking, temporal logic, simulation, induction.

1. Introduction

Verification plays an important role in designing reliable computer systems. With the increasing involvement of computer hardware and software in daily life, checking the safety and correctness of these systems has become essential and sometimes even critical. Therefore it is an imperative task for computer scientists to develop advanced verification techniques that will support the development of reliable systems.

Two main approaches to program verification are testing and formal verification.

[1]This paper is supported by the grant RFBR 06-01-00106 and INTAS 05-1000008-8144.

Testing assumes an activity of generating a collection of input data (test cases), running system to be verified on these data, and then analyzing its behavior. 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 assumes that one provides a mathematical model for the system under study, specifies the properties the system should comply with, and then applies mathematical techniques (deductive methods, model-checking, equivalence-checking, etc.) to check that the model satisfies the properties. Formal verification is relatively inexpensive in comparison to exhaustive simulation and can be applied at various points through the development process. It receives an ample algorithmic support from various branches of mathematics and manifests its strength in areas where other verification methods are inadequate. That is why formal verification is now becoming an indispensable stage of software development process.

According to the approach towards formalization, formal methods fall into the following major categories: model checking, theorem proving, and equivalence checking. Model checking is technique which allows verification of computer system by checking that a model $M(P)$ (represented as transition system derived from hardware or software design P) satisfies a formal specification φ (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, including tableau-based, symbolic and on-the-fly algorithms, partial order reduction techniques and many others (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) $M(P)$.

It is quite another matter of checking infinite state models or infinite families of parameterized systems $\mathcal{F} = \{P_k\}_{k=1}^{\infty}$. The latter is one of the most challenging problems in verification today. As a matter of fact, 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 $p_1\|p_2\|\dots\|p_k\|q$ of similar "user" processes $p_1, p_2, \dots, p_k$ 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 $P_k = p_1\|p_2\|\dots\|p_k\|q$ and a temporal formula φ, check that each transition system $M(P_k)$ satisfies φ.

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.

Symmetry is commonly used to avoid the combinatorial explosion in the number of states in transition systems. Suppose that a transition system P has a non-trivial group G of permutations acting on the state set S and preserving 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 $P, s \models \varphi \iff P_G, \theta(s) \models \varphi$. 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.

Symmetry can be also exploited to reduce the number of cases to be checked when a specification of a system is also parameterized (say, when $\varphi_k = \bigwedge_{i=1}^{k} \psi[i]$). In [15] it was demonstrated that if a group G of permutations acting on the state set S of a transition system P offers some nice properties then the verification problem $P_k \models \varphi_k$ may be reduced to that of checking $P_k \models \psi[1]$.

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 some 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 appro-

priately, one gets an abstract model $h(P)$. If a temporal formula φ to be checked is built of formulae from A then $h(P) \models \varphi \Longrightarrow P \models \varphi$. A theoretical framework for abstraction technique has been developed in [5, 12, 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 $Form$ 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 $P_k = p_1 \| p_2 \| \dots \| p_k \| q$, find a finite transition system $\mathcal{I}$ such that

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

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 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 traditional model-checking techniques for finite state transition systems. This approach to the verification of parameterized networks was introduced in [22, 26] and developed in many papers (see [4] for a survey).

The central problem with induction technique is 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 was applied in [7, 8, 15] 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 parameterized 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 conventional 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. In [7, 8] it was 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 [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 preoreders 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 succeed. 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 in [21] a block simulation preorder (which is an amalgamation of block bisimulation [15] and visible simulation [18]) and showed that by using this preorder one can generate invariants of some parameterized systems straightforwardly.

In this paper we continue this line of research. Unfortunately, asynchronous composition of processes is not monotonic w.r.t. block simulation in general case. Therefore 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 in the induction-based verification techniques. To demonstrate the feasibility of quasi-block simulation we consider Dijkstra's token ring algorithm [13]. This algorithm was treated as a case study in [8]. We demonstrate that its induction-based verification 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. In Section 4 we apply induction technique based on quasi-block simulation to token ring protocol. 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 actions, not containing the distinguished action τ (invisible action),*
- *$R \subseteq S \times A \cup \{\tau\} \times S$ 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.*

Any triple (s, a, t) from R is called a *transition*. To simplify notation we write $s \xrightarrow{a}_M t$ instead of $(s, a, t) \in R$ and often elide the subscript M when a specific LTS is assumed. A *finite path* π of LTS M is a finite sequence $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots \xrightarrow{a_{k-1}} s_k$ of transitions $(s_i \xrightarrow{a_i} s_{i+1})$. The length $|\pi|$ of a finite path π is the number of states this path passes through, i.e. $|\pi| = k$. An *infinite path* π is an infinite sequence of transitions $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots \xrightarrow{a_{j-1}} s_j \xrightarrow{a_j} \dots$ $(s_i \xrightarrow{a_i} s_{i+1})$. We write $\pi[i]$ for the state s_i of a path π.

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

Syntax of CTL*. Let Σ be a set of propositional variables. CTL* formulas are defined using state formulas and auxilary path formulas.

State formulas are defined as follows:

- For any $p \in \Sigma$ propositional variable p is a state formula.
- If φ and ψ are state formulas, then $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ are also state formulas.
- If φ is a path formula, then $\mathsf{E}\varphi$ and $\mathsf{A}\varphi$ are state formulas.

Path formulas are defined as follows:

- If φ is a state formula, then φ is a path formula.
- If φ and ψ are path formulas, then $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ are also path formulas.
- If φ and ψ are state formulas, then $\mathsf{X}\varphi$, $\mathsf{F}\varphi$, $\mathsf{G}\varphi$, $\varphi\mathsf{U}\psi$ are path formulas.

Any state formula is a formula of Full Branching Time Logic CTL*.

Semantics of CTL*. Semantics of CTL* may be defined on Labelled Transition Systems. Let $M = \langle S, S_0, A, R, \Sigma, L\rangle$ be a LTS. We write $M, s \models \varphi$ to mean that formula φ (φ is a state or path formula) is true in state s of LTS M. Also we write $M, \pi \models \varphi$ to mean that formula φ (φ is a state or path formula) is true on path π of LTS M.

Then, semantics of CTL* is defined as follows:

- $M, s \models p \Leftrightarrow p \in L(s)$.
- $M, s \models \neg\varphi \Leftrightarrow \neg(M, s \models \varphi)$.
- $M, s \models \varphi \wedge \psi \Leftrightarrow M, s \models \varphi$ and $M, s \models \psi$.
- $M, s \models \varphi \vee \psi \Leftrightarrow M, s \models \varphi$ or $M, s \models \psi$.
- $M, s \models \mathsf{E}\varphi \Leftrightarrow$ there is a path in π from state s in LTS M such that $M, \pi \models \varphi$.
- $M, s \models \mathsf{A}\varphi \Leftrightarrow$ for any path π from state s in LTS M it holds $M, \pi \models \varphi$.
- $M, \pi \models \varphi \Leftrightarrow$ in the first state s of path π it holds $M, s \models \varphi$.
- $M, \pi \models \neg\varphi \Leftrightarrow \neg(M, \pi \models \varphi)$.
- $M, \pi \models \varphi \wedge \psi \Leftrightarrow M, \pi \models \varphi$ and $M, \pi \models \psi$.

- $M, \pi \models \varphi \vee \psi \Leftrightarrow M, \pi \models \varphi$ or $M, \pi \models \psi$.
- $M, \pi \models \mathsf{X}\varphi \Leftrightarrow M, \pi^1 \models \varphi$.
- $M, \pi \models \mathsf{F}\varphi \Leftrightarrow$ there is $k \geq 0$ such that $M, \pi^k \models \varphi$.
- $M, \pi \models \mathsf{G}\varphi \Leftrightarrow$ for any $i \geq 0$ it holds $M, \pi^i \models \varphi$.
- $M, \pi \models \varphi\mathsf{U}\psi \Leftrightarrow$ there is $k \geq 0$ such that $M, \pi^k \models \psi$ and for any $0 \leq i < k$ it holds $M, \pi^i \models \phi$.

Formula φ is true in LTS $M = \langle S, S_0, A, R, \Sigma, L\rangle$ ($M \models \varphi$ in symbols), if for any initial state $s_0 \in S_0$ it holds $M, s_0 \models \varphi$.

Formulas of ACTL*-X are formulas of CTL* in positive normal form, without subformulas $\mathsf{E}\varphi$ and $\mathsf{X}\psi$.

Let M_1 and M_2 be two LTSs, $M_1 = \left\langle S^1, S^1_0, A^1, R^1, \Sigma^1, L^1\right\rangle$ and $M_2 = \left\langle S^2, S^2_0, A^2, R^2, \Sigma^2, L^2\right\rangle$, $\Sigma^1 \cap \Sigma^2 = \emptyset$. We call a *synchronizer* any pair $\Gamma = \langle \Delta, ^{-}\rangle$, where $\Delta \subseteq A^1$, and $^{-} : \Delta \rightarrow A^2$ is an injection, relating 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 some actions a of one LTS are executed only synchronously with the co-actions $\overline{a}$ of another LTS. Thus, a pair $(a, \overline{a})$ 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 $\overline{a}$) 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 Γ is an LTS $M = M_1 \parallel_\Gamma M_2 = \langle S, S_0, A, R, \Sigma, L\rangle$ such that*

- $S = S^1 \times S^2$, $S_0 = S^1_0 \times S^2_0$, $A = A^1 \cup A^2 \setminus (\Delta \cup \overline{\Delta})$, $\Sigma = \Sigma^1 \cup \Sigma^2$, $L(s, u) = L^1(s) \cup L^2(u)$
- *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:*
  - *$a \in A^1 \setminus \Delta$, $u = v$, $(s, a, t) \in R^1$ (M_1 executes a),*
  - *$a \in A^2 \setminus \overline{\Delta}$, $s = t$, $(u, a, v) \in R^2$ (M_2 executes a),*
  - *$a = \tau$, and there exists $b \in \Delta$ such that $(s, b, t) \in R^1$ and $(u, \overline{b}, v) \in R^2$ (M_1 and M_2 communicate),*

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 of M that either are visible (i.e. marked with an action $a \neq \tau$) or affect the basic propositions of φ:

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

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

Definition 3. *A* finite block *from a state s_1 w.r.t. a set of events E is a 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 $i : 1 \leq i < m$. 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 $MAXF(E, s)$ and $MAXI(E, s)$ 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 be the LTS $M = M_1 \parallel_\Gamma \cdots \parallel_\Gamma M_n$ and δ be the path $\delta = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{i-1}} s_i \xrightarrow{a_i} \ldots$ in M. We define the projection $\pi = pr_{M_l}(\delta)$ on the model M_l by induction on the length of path.*

If $n = 1$ then $\pi_1 = s_1$.

If $n = k \geq 2$ then the following cases may occur:

- *Model M_l does not make a move in $(s^1_{k-1}, \ldots, s^n_{k-1}) \xrightarrow{a_k} (s^1_k, \ldots, s^n_k)$. In this case we put $\delta_k = \delta_{k-1}$.*
- *The transition $(s^1_{k-1}, \ldots, s^n_{k-1}) \xrightarrow{a_k} (s^1_k, \ldots, s^n_k)$ is a local move of M_l. In this case $\delta_k = \delta_{k-1} \xrightarrow{a_k} s^l_k$.*
- *The transition $(s^1_{k-1}, \ldots, s^n_{k-1}) \xrightarrow{a_k} (s^1_k, \ldots, s^n_k)$ is a synchronous move of M_l and M_j ($a_k = \tau$). This transition is built as a composition of some transitions $s^l_{k-1} \xrightarrow{b} s^l_k$ and $s^j_{k-1} \xrightarrow{\overline{b}} s^j_k$. In this case $\delta_k = \delta_{k-1} \xrightarrow{b} s^l_k$.*

Let $prb_{M_l}(\delta)$ (block projection) denote any finite or infinite block B such that B starts with $pr_{M_l}(\delta)$.

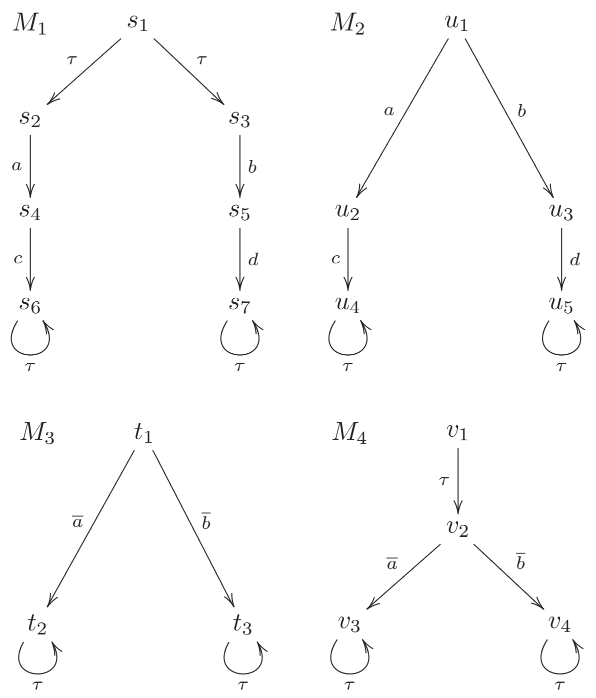
Definition 5. *Let $M^i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i\rangle$, $i = 1, 2$, be two LTSs, let Σ_0 be a subset of $\Sigma^1 \cap \Sigma^2$, and let E^1 and E^2 be some sets of events of M^1 and M^2. Then a binary relation $H \subseteq S^1 \times S^2$ is called a* quasi-block simulation *on M^1 and M^2 w.r.t. Σ_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 $B' = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{a} s_{m+1} \in MAXF(E^1, s_1)$ there is a block $B'' = t_1 \xrightarrow{\tau} t_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{a} t_{n+1} \in MAXF(E^2, t_1)$ such that $(s_{m+1}, t_{n+1}) \in H$, and $(s_i, t_j) \in H$ holds for every pair i, j, $1 \leq i \leq m$, $1 \leq j \leq n$.*
3. *For every infinite block $B' = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_m \xrightarrow{\tau} \cdots \in MAXI(E^1, s_1)$ there is an infinite block $B'' = t_1 \xrightarrow{\tau} t_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_n \xrightarrow{\tau} \cdots \in MAXI(E^2, t_1)$, such that $(s_i, t_j) \in H$ holds for every pair i, j, $1 \leq i$, $1 \leq j$.*

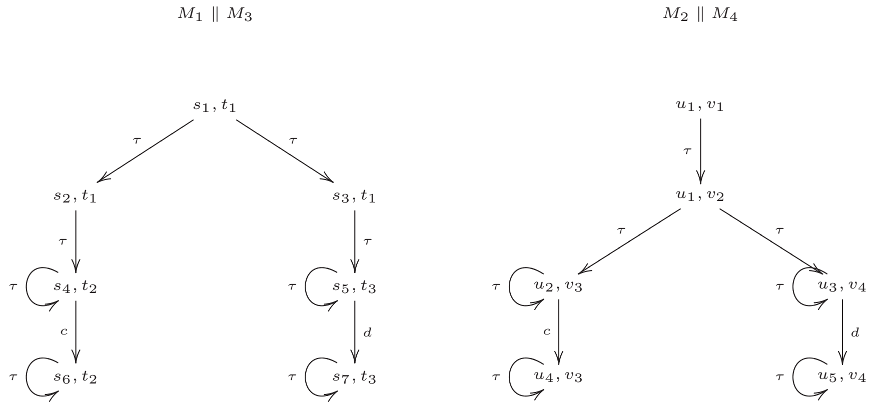
We write $M^1 \preceq^{qb}_{\Sigma_0} M^2$ iff there exist two sets of events E^1 and E^2 of LTSs M^1 and M^2 and a binary relation $H \subseteq S^1 \times S^2$ such that H is a quasi-block simulation on M^1 and M^2 w.r.t. Σ_0, E^1, E^2, and for every initial state $s_0 \in S^1_0$ there exists an initial state $t_0 \in S^2_0$ such that $(s_0, t_0) \in H$. If both E^1 and E^2 are well-formed w.r.t. φ then we say that the quasi-block simulation $M^1 \preceq^{qb}_{\Sigma_\varphi} M^2$ is also well-formed w.r.t. φ. A *block simulation* ($M^1 \preceq^{b}_{\Sigma_0} M^2$ in symbols) is a quasi-block simulation w.r.t. Σ_0, $Observ(M^1, \Sigma_0)$, $Observ(M^2, \Sigma_0)$.

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 [25]. 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 1. Let M_1, M_2, M_3, M_4 be LTSs depicted below and $\Gamma = (\{\{a, b\}, \{a \rightarrow \overline{a}, b \rightarrow \overline{b}\}\})$ be a synchronizer.



It can be seen that $M_1 \preceq^b_{\emptyset} M_2$ and $M_3 \preceq^b_{\emptyset} M_4$. However, there exists no block simulation for the compositions $M_1 \parallel_\Gamma M_3$ and $M_2 \parallel_\Gamma M_4$.



3. The basic features of quasi-block simulation

As it can be seen from the definitions above $M^1 \preceq^b_{\Sigma_0} M^2 \implies M^1 \preceq^{qb}_{\Sigma_0} M^2$. Moreover, quasi-block simulation can be reduced to block simulation. Let $M^i = \langle S^1, S^1_0, A^1, R^1, \Sigma^1, L^1 \rangle$, $i = 1, 2$, be two LTSs such that $M^1 \preceq^{qb}_{\Sigma_0} M^2$ w.r.t. sets of events E^1 and E^2. Consider an auxiliary visible action ε such that $\varepsilon \notin A^1 \cup A^2$, and build the LTSs $\widetilde{M}^i = \left\langle S^i, S^i_0, A^i \cup \{\varepsilon\}, \widetilde{R}^i, \Sigma^i, L^i \right\rangle$, $i = 1, 2$, 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 included in the sets of events E^1 and E^2.

Theorem 1. $M^1 \preceq^{qb}_{\Sigma_0} M^2 \iff \widetilde{M}^1 \preceq^b_{\Sigma_0} \widetilde{M}^2$.

Theorem 1 may have a considerable utility in checking quasi-block 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 $M^i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i \rangle$, $i = 1, 2, 3, 4$, be four LTS's such that*

- $(\Sigma^1 \cup \Sigma^2) \cap (\Sigma^3 \cup \Sigma^4) = \emptyset$*,*
- $A^1 = A^2 = A'$*,* $A^3 = A^4 = A''$*, and* $A' \cap A'' = \emptyset$*.*

Let Σ' and Σ'' be the 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 $M_3 \preceq^{qb}_{\Sigma''} M_4$ implies $M_1 \parallel_\Gamma M_3 \preceq^{qb}_{\Sigma' \cup \Sigma''} M_2 \parallel_\Gamma M_4$.

Proof. Let H' and H'' be the relation of quasi-block simulation on M_1, M_2 w.r.t. Σ', $Event^1$, $Event^2$ and M_3, M_4 w.r.t Σ'', $Event^3$, $Event^4$ respectively. We build such a relation $H \subseteq (S^1 \times S^3) \times (S^2 \times S^4)$, 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 show that H is a quasi-block simulation of M_{13} and M_{24} w.r.t. some $Event'$ and $Event''$.

$Event'$ is built as follows.

$((s^1, s^3), a, (t^1, t^3)) \in Event'$ iff one of the conditions met:

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

$Event''$ is built similar to $Event'$.

The proof idea is as follows. Block simulation may fail for the composition of models only in case of failure of condition 2. Condition 2 is failed only if blocks expand in comparison to that in original models. We build $Event'$ and $Event''$ in such a manner that blocks are not enlarged. Conditions 1-3 are proved by considering all the possible ways of the composition of original blocks.

We show for any pair of states $((s^1, s^3), (s^2, s^4)) \in H$ that conditions of quasi-block simulation are satisfied.

1. $L((s^1, s^3)) \cap (\Sigma' \cup \Sigma'') = (L(s^1) \cup L(s^3)) \cap (\Sigma' \cup \Sigma'') = L(s^1) \cap \Sigma' \cup L(s^3) \cap \Sigma'' = L(s^2) \cap \Sigma' \cup L(s^4) \cap \Sigma'' = (L(s^2) \cup L(s^4)) \cap (\Sigma' \cup \Sigma'') = L((s^2, s^4)) \cap (\Sigma' \cup \Sigma'')$.
2. Let B_{13} be any block such that $B_{13} \in MAXF(Event', (s^1, s^3))$. $B_{13} = (s^1_1, s^3_1) \xrightarrow{\tau} (s^1_2, s^3_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^1_m, s^3_m) \xrightarrow{a} (s^1_{m+1}, s^3_{m+1})$, $(s^1_1, s^3_1) = (s^1, s^3)$.

   Let $B_1 = prb_{M_1}(B_{13})$, $B_3 = prb_{M_3}(B_{13})$.

   There are several cases to be considered:

   (a) $a = \tau$ is a result of synchronous move of some $s^1_m \xrightarrow{b} s^1_{m+1}$ and $s^3_m \xrightarrow{\bar{b}} s^3_{m+1}$ in M_1 and M_3 correspondingly.

   As $(s^1, s^2) \in H_{12}$, $(s^3, s^4) \in H_{34}$, then there is the blocks B_2 and B_4 such that $B_2 \in MAXF(Event^2, s^2)$, B_2 matches B_1, and $B_4 \in MAXF(Event^4, s^4)$, B_4 matches B_3.

   B_{34} is built by shuffling all the transitions of B_3 and B_4 excluding the last ones. The parallel composition of the last transitions of B_3 and B_4 form the last transition of B_{34}.

   (b) B_{13} contains only the part of B_3. The blocks B_1 and B_3 are finite ones.

   As $(s^1, s^2) \in H_{12}$, $(s^3, s^4) \in H_{34}$, then there is blocks B_2 and B_4 such that $B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{a} s^2_{k+1} \in MAXF(Event^2, s^2)$, B_2 matches B_1, and $B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_l \xrightarrow{b} s^4_{l+1} \in MAXF(Event^4, s^4)$, B_4 matches B_3.

   B_{24} is constructed as $B_{24} = (s^2_1, s^4_1) \xrightarrow{\tau} (s^2_1, s^4_2) \xrightarrow{\tau} (s^2_2, s^4_2) \xrightarrow{\tau} \ldots \xrightarrow{\tau} (s^2_k, s^4_2) \xrightarrow{a} (s^2_{k+1}, s^4_2)$. The block B_{24} matches to B_{13}.

   (c) B_{13} contains only the part of B_3. The block B_1 is a finite one and B_3 is an infinite one.

   As $(s^1, s^2) \in H_{12}$, $(s^3, s^4) \in H_{34}$, then there is blocks B_2 and B_4 such that $B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{a} s^2_{k+1} \in MAXF(Event^2, s^2)$, B_2 matches B_1, and $B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_l \xrightarrow{b} s^4_{l+1} \in MAXF(Event^4, s^4)$, B_4 matches B_3.

   As in the previous case we construct B_{24} as $B_{24} = (s^2_1, s^4_1) \xrightarrow{\tau} (s^2_1, s^4_2) \xrightarrow{\tau} (s^2_2, s^4_2) \xrightarrow{\tau} \ldots \xrightarrow{\tau} (s^2_k, s^4_2) \xrightarrow{a} (s^2_{k+1}, s^4_2)$. The block B_{24} matches B_{13}.

   (d) The cases when B_3 is fully included in B_{13} and B_1 is only partly included in B_{13} are similar to the previous two cases.

3. Let B_{13} be an infinite block such that $B_{13} \in MAXI(Event', (s^1, s^3))$. $B_{13} = (s^1_1, s^3_1) \xrightarrow{\tau} (s^1_2, s^3_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (s^1_m, s^3_m) \xrightarrow{a} (s^1_{m+1}, s^3_{m+1})$, $(s^1_1, s^3_1) = (s^1, s^3)$.

   Let $B_1 = prb_{M_1}(B_{13})$, $B_3 = prb_{M_3}(B_{13})$.

   We have to consider several cases:

   (a) The blocks B_1 and B_3 are infinite ones.

   As $(s^1, s^2) \in H_{12}$, $(s^3, s^4) \in H_{34}$, then there is blocks B_2 and B_4 such that $B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{a} s^2_{k+1} \in MAXI(Event^2, s^2)$, B_2 matches B_1, and $B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_l \xrightarrow{b} s^4_{l+1} \in MAXI(Event^4, s^4)$, B_4 matches B_3.

   B_{34} is built by shuffling all the transitions of B_3 and B_4.

   (b) The block B_1 is an infinite block and the block B_3 is a finite one.

   As $(s^1, s^2) \in H_{12}$, $(s^3, s^4) \in H_{34}$, then there is blocks B_2 and B_4 such that $B_2 = s^2_1 \xrightarrow{\tau} s^2_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^2_k \xrightarrow{a} s^2_{k+1} \in MAXI(Event^2, s^2)$, B_2 matches B_1, and $B_4 = s^4_1 \xrightarrow{\tau} s^4_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s^4_l \xrightarrow{b} s^4_{l+1} \in MAXI(Event^4, s^4)$, B_4 matches B_3.

   B_{34} is built by shuffling all the transitions of B_3 and B_4.

   (c) The case when the block B_1 is a finite one and the block B_3 is an infinite one is similar to the previous case.

□

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 $M^i = \langle S^i, S^i_0, A^i, R^i, \Sigma^i, L^i \rangle$, $i = 1, 2$, 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. Σ_0 iff every pair $(s', s'') \in H$ comply with the following requirements:

1. $L^1(s') \cap \Sigma_0 = L^2(s'') \cap \Sigma_0$.
2. For every path π', $\pi' = {s'}_1 \xrightarrow{a_1} {s'}_2 \xrightarrow{a_2} \dots \xrightarrow{a_{k-1}} {s'}_k \xrightarrow{a_k} \dots$, $s'_0 = s'$ there is a path π'', $\pi'' = {s''}_1 \xrightarrow{a_1} {s''}_2 \xrightarrow{a_2} \dots \xrightarrow{a_{k-1}} {s''}_k \xrightarrow{a_k} \dots$, $s''_0 = s''$ and partitions $P'_1 P'_2 \dots$, $P''_1 P''_2 \dots$ 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 $M^1 \preceq^{st}_{\Sigma_0} M^2$ to indicate the existence of stuttering simulation between M^1 and M^2.

Theorem 3. $M^1 \preceq^{qb}_{\Sigma_0} M^2 \Longrightarrow M^1 \preceq^{st}_{\Sigma_0} M^2$.

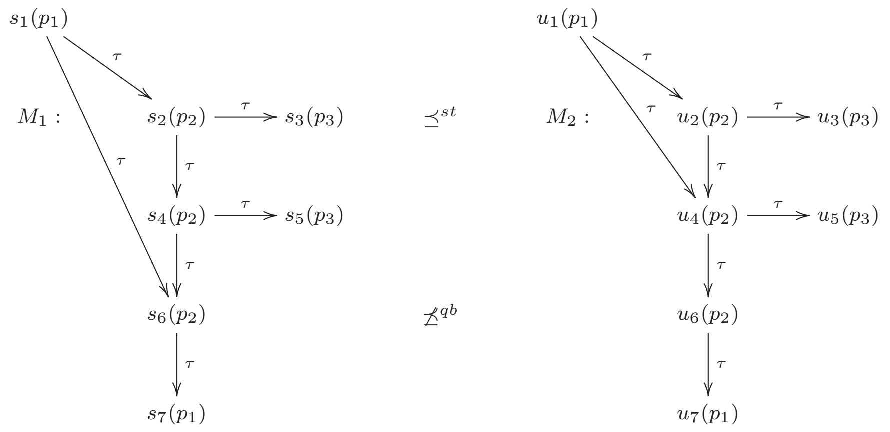
A proof this theorem is straightforward.

Since stuttering simulation preserves the satisfiability of temporal formulae from $ACTL_{-X}$, Theorem 3 brings us to the following conclusion.

Theorem 4. *Suppose that a quasi-block 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 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 quasi-block one.

Example 2. Consider models M_1 and M_2 below. The evaluation functions associate with every state exactly one of basic propositions from the set $\Sigma_0 = \{p_1, p_2, p_3\}$. It could be directly checked that $M_1 \preceq^{st}_{\Sigma_0} M_2$.


$$\begin{tikzcd}
s_1(p_1) \arrow[dr, "\tau"] \arrow[dddr, "\tau"'] & & & & u_1(p_1) \arrow[dr, "\tau"] \arrow[ddr, "\tau"'] & & \\
M_1: & s_2(p_2) \arrow[r, "\tau"] \arrow[d, "\tau"] & s_3(p_3) & \preceq^{st} & M_2: & u_2(p_2) \arrow[r, "\tau"] \arrow[d, "\tau"] & u_3(p_3) \\
 & s_4(p_2) \arrow[r, "\tau"] \arrow[d, "\tau"] & s_5(p_3) & & & u_4(p_2) \arrow[r, "\tau"] \arrow[d, "\tau"] & u_5(p_3) \\
 & s_6(p_2) \arrow[d, "\tau"] & & \not\preceq^{qb} & & u_6(p_2) \arrow[d, "\tau"] & \\
 & s_7(p_1) & & & & u_7(p_1) &
\end{tikzcd}$$

Suppose that there exists a quasi-block 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 $u_4 \xrightarrow{u}_6$ 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, $u_4 \xrightarrow{u}_6$ has to be in E_2. This certifies that $M_1 \not\preceq^{qb}_{\Sigma_0} M_2$.

The fact that quasi-block 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. They demonstrated 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. The feasibility of this approach has been demonstrated by applying it to the verification of Dijkstra's token ring algorithm. We extend this approach by replacing strong simulation with quasi-block simulation. This makes it possible to get rid of abstraction h and considerably simplify the verification algorithm.

4.1. Dijkstra's token ring algorithm

In Dijkstra's token ring algorithm a network is composed of similar processes connected in bidirectional ring. A token t is passed in the clockwise direction. Each process may send a request the token (a signal s) in the counter-clockwise direction. A process sends the signal s if either it intends acquiring the token, or it receives the signal from its neighbor on the counter-clockwise side.

A state of each process is described by a word XYZ, where

- X stands for a control state of the process, $X \in \{n, d, c\}$; state symbols n, d, and c stand for the neutral state, the delayed state, and the critical section respectively.
- Y indicates an intention of the process to acquire the token, $Y \in \{b, w\}$; symbols b and w stand for black (there is a process to the right that wishes to acquire the token) and white (no one to the right is interested in the token) respectively.
- Z indicates whether the token is granted to the process or not, $Z \in \{t, e\}$; t means that the process has a token, and e means that the process is empty.

The transition relation of each such process is depicted on the table 1. Numbers 1, 2, and 3 are auxiliary counter values. The star symbol admits any allowed element in

Table 1: Transitions table of Dijkstra's algorithm

1new $\xrightarrow{rcvtok}$ 1ntw	1new $\xrightarrow{rcvsig}$ 2neb	1ntw $\xrightarrow{rcvsig}$ 3new
1ntw $\xrightarrow{\tau}$ 1ctw	1neb $\xrightarrow{rcvtok}$ 3new	1new $\xrightarrow{\tau}$ 2deb
1neb $\xrightarrow{\tau}$ 1deb	1dew $\xrightarrow{rcvsig}$ 1deb	1de* $\xrightarrow{rcvtok}$ 1ct-
1ctw $\xrightarrow{rcvsig}$ 1ctb	1ctw $\xrightarrow{\tau}$ 1ntw	1ctb $\xrightarrow{\tau}$ 1ntw
1ctw $\xrightarrow{rcvsig}$ 1ctb	2*** $\xrightarrow{sndsig}$ 1—	2n** $\xrightarrow{rcvtok}$ 3—w
2d** $\xrightarrow{rcvtok}$ 1ct-	3*** $\xrightarrow{sndtok}$ 1—	

the position. The dash symbol means that the values of the corresponding element before and after transition are the same.

Let P_0 be a distinguished process with $1wnt$ as the initial state. All other processes P_i, $1 \leq i \leq n$, have $1wne$ as the initial state. Each process P_i, $0 \leq i \leq n$, has the set of actions $\{rcvsig_i, sndsig_i, rcvtok_i, sndtok_i\}$, and these sets are pairwise disjoint. Every process P_i is synchronized with the P_{i+1} (modulo n) by the synchronizer $\Gamma_i = \{\{rcvsig_i, sndtok_i\}, \{rcvsig_i \rightarrow sndsig_{i+1}, sndtok_i \rightarrow rcvtok_{i+1}\}$. Thus we obtain the infinite family of parameterized systems $\mathcal{F} = \{P^k\}_{k=1}^{\infty}$ such that $P^k = P_0 \| P_1 \dots P_k$. This parameterized network can be described in terms of network diagrams as follows:

$$\begin{aligned} &S \rightarrow P_0 \parallel A \\ &A \rightarrow P \parallel A \\ &A \rightarrow P \parallel P \parallel P \end{aligned}$$

We have implemented in Python a straightforward algorithm to check block simulation of LTSs and apply to the network described above. The results are shown in table 2.

Since it was found that $h(P^4) \preceq^{qb} h(P^3)$, Theorems 1, 2, and 4 guarantee that $h(P^3)$ is an invariant of the family $\mathcal{F}$. Unfortunately, abstraction techniques should be also applied in case of Dijkstra's algorithm. This experiment lends support to the feasibility of the using of quasi-block simulation in the framework of invariant-based verification technique.

Table 2: Computation of block simulation. Results.

LTSs	time	have block simulation
P^3 and P^2	0.99s	no
P^4 and P^3	17.8s	no
abstract P^3 and P^2	0.2s	no
abstract P^4 and P^3	0.22s	yes

4.2. Tree wave algorithm with neighbor synchronisation

We consider an algorithm in which processes are organized into a binary tree. The root of a tree sends message to children. Each intermediate tree node executes synchronous action *sync_neighbor* (which is synchronized with its sibling) and sends message to the children. Any leaf node sends a message back on receipt of the message from the leaf's parent. Then, intermediate nodes pass messages freely from the lower nodes to the upper ones.

It may be useful to check the property that the root node eventually receives its message back after it has sent a message to the children. This property is satisfied on the model with root, two intermediate nodes and four leaves. To check the property on the infinite family of models, we distinguish states of the root nodes as visible and states of other processes as invisible ones.

There exists a block simulation of the model with two intermediate nodes and four leaves by the model with two leaves. Thus, we can infer that there exists a quasi-block simulation of any tree by the tree with two leaves. Note that a block simulation may be unavailable due to the properties of block simulation.

This example does not deal with any property-specific abstraction. After showing that any model of the family is simulated by the invariant (the tree with two leaves) it is possible to check any property on visible variables, i.e. any property of the root node.

5. 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 practical case studies that

could indicate convincingly the advisability of using quasi-block simulation in the verification of parameterized systems. It depends to a large extent also on how much effectively quasi-block simulation can be checked. We assume that Theorem 1 could give an essential prerequisite for constructing efficient checking procedures.

References

[1] Apt K.R., Kozen D. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986, p. 307–309. 38

[2] Bosnacki D., Dams D., Holenderski L. A heuristic for symmetry reductions with scalarset. In *Procedings of FME2001*, Lecture Notes in Computer Science, 2021, 2001, p. 518–533. 39

[3] Brown M.C., Clarke E.M., Grumberg O. Characterizing finite Kripke structures in propositional temporal logics. *Theoretical Computer Science*, v. 59, 1988, p. 115–131. 46, 51

[4] Calder M., Miller A. Five ways to use induction and symmetry in the verification of networks of processes by model-checking. in *Proceedings of AvoCS 2002 (Automated Verification of Critical Systems)*, 2002, p. 29–42 39, 40, 52

[5] Clarke E.M., Grumberg O., Long D.E. Model checking and abstraction. In *Proceedings of Principles of Programming Languages*, 1992. 40

[6] Clarke E.M., Filkorn T., Jha S. Exploiting symmetry in temporal logic model checking. In *Proceedings of CAV'93*, Lecture Notes in Computer Science,697, 1993, p. 450–461. 39

[7] Clarke E.M., Grumberg, O., and Jha, S. Verifying parameterized networks using abstraction and regular languages, In *Proceedings of the 6-th International Conference on Concurrency Theory*, 1995. 40, 41

[8] Clarke E.M., Grumberg, O., and Jha, S. Verifying parameterized networks. *ACM Transactions on Programming Languages and Systems*, 19(5), 1997, p. 726–750. 40, 41, 53

[9] Clarke E.M., Grumberg O., Peled D.A. Model checking. MIT Press, 1999. 38, 42, 51

[10] Clarke E., Talupur M., Touili T., Veith H. Verification by network decomposition. In *Proceedings of CONCUR'04*, Lecture Notes in Computer Science, 3170, 2004, p. 276–291. 52

[11] Cleaveland R., Sokolsky O. Equivalence and Preorder Checking for Finite-State Systems, In *Handbook of Process Algebra*, Elsevier, 2001, p. 391–424. 48

[12] Dams D., Grumberg O., Gerth R. Abstract interpretation of reactive systems: abstractions preserving $ACTL^*$, $ECTL^*$ and CTL^*. In *IFIP Working Conference and Programming Concepts, Methods and Calculii*, 1994. 40

[13] Dijkstra E. Invariance and non-determinacy. In *Proceedings of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*, 1985, p. 157–165. 41

[14] Donaldson A.F. Miller A. Automatic symmetry detection for model checking using computational group theory. In *Proceedings of the 13th International Symposium on Formal Methods Europe (FME 2005)*, Lecture Notes in Computer Science, 3582, 2005, p. 481–496. 39

[15] Emerson E.A., Namjoshi K.S. Reasoning about rings. In *Proceedings 22th ACM Conf. on Principles of Programming Languages, POPL'95*, 1995, p.85–94. 39, 40, 41, 46, 52

[16] Emerson E.A., Sistla A.P. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2), 1996, p.105–131. 39

[17] Etessami K., Schuller R., Wilke T. Fair simulation relations, parity games, and state space reduction for Buchi automata. In *Proceedings of 28th International Collquium "Automata, Languages and Programming"*, Lecture Notes in Computer Science, 2076, 2001, p. 694–707. 48

[18] Gerth R., Kuiper R., Peled D., Penczek W. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2), 1999, p.132–152. 41, 51

[19] Ip C.N., Dill D.L. Verifing systems with replicating components in murϕ. *Formal Methods in System Design*, 14, 1999, p.273–310. 39

[20] Kesten Y., Pnueli A. Verification by finitary abstraction. *Information and Computation*, 163, 2000, p.203–243. 40

[21] Konnov I.V., Zakharov V.A. An approach to the verification of symmetric parameterized distributed systems. *Programming and Computer Software* 31(5), 2005, p. 225–236. 41

[22] Kurshan R.P., MacMillan K.L. Structural induction theorem for processes. In *Proceedings of the 8-th International Symposium on Principles of Distributed Computing, PODC'89*, 1989, p. 239–247. 40

[23] Lesens D., Saidi H. Automatic verification of parameterized networks of processes by abstraction. In *Proceedings of the 2-nd International Workshop on the Verification of Infinite State Systems (INFINITY'97)*, 1997. 40

[24] Manku G.S., Hojati R., Brayton R.K. Structural symmetries and model checking. In *Proceedings of International Conference on Computer-Aided Verification (CAV'98)*, 1998, p. 159–171. 39

[25] Penczek W., Gerth R., Kuiper R., Szreter M. Partial order reductions preserving simulations. In *Proceedings of the CSP'99 Workshop*, 1999, p. 153–171. 46

[26] Wolper P., Lovinfosse. Properties of large sets of processes with network invariants. Lecture Notes in Computer Science, 407, 1989, p. 68–80. 40