# Computing Testing Equivalence with Binary Decision Diagrams *

Robert Meolic, Tatjana Kapus, Zmago Brezočnik
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17, SI-2000 Maribor, Slovenia
{meolic,kapus,brezocnik}@uni-mb.si

**Abstract**

*Process algebras are a convenient tool for describing and reasoning about the behaviour of concurrent systems. A variety of equivalence relations are used to study the relationship between different systems or between different levels of abstractions of the system. This paper reports on an implementation of a testing equivalence with binary decision diagrams (BDDs). It is defined in terms of observations that process may or must satisfy. As the testing equivalence is shown to be an instance of a bisimulation equivalence, efficient algorithms for computing bisimulations using BDDs could be employed for its implementation.*

## 1 Introduction

A basic problem in automatic verification of systems is to prove that the proposed implementation satisfies its specification. Many relationships between systems have been studied as being suitable for solving this problem, among them some different types of equivalences. The most popular, trace equivalences and bisimulation equivalences, are for many problems not enough or too restrictive. On the contrary, testing equivalences seem to be a convenient approach.

The work presented relies on process algebras. There, systems are described as labelled transition systems called processes [6]. Many operators are defined for building and manipulating processes, e.g. process compositions and abstractions. Different kinds of process algebras exist. They all enable the usage of nondeterminism, concurrency, and hiearchy in description. Process algebras are rapidly becoming one of the most important mathematical tools used for automatic verification due to efficient implementation of algorithms with BDDs [1, 6].

Further in this paper we first define a process and the testing equivalence we use. In Section 3 we introduce bisimulations and describe how to use them for checking the testing equivalence. In Section 4 we present algorithms for the implementation of testing equivalence with BDDs. We conclude with some directions for future work.

*Work supported by Ministry of Science and Technology, Republic of Slovenia.

## 2 Definition of Testing Equivalence

A simple view of processes is that they are devices which transit between their internal states by performing actions. The actions can be visible to an external observer or not. In the last case the action is called silent action. More formally, a process is defined as follows.

**Definition 1.** A process $P$ is a 4-tuple $(\mathcal{S}_P, Act, \delta_P, p_0)$, where:

- $\mathcal{S}_P$ is a set of process states,
- $Act$ is a set of actions containing silent action $\tau$,
- $\delta_P \subseteq \mathcal{S}_P \times Act \times \mathcal{S}_P$ is the transition relation, and
- $p_0$ is initial state.

Every triple $(p, \alpha, p') \in \delta_P$ determines a transition of the process from state $p$ to $p'$ performing action $\alpha$. In this case, $p'$ is said to be an $\alpha$-derivate of $p$. A state $p$ is a *deadlock state* if process can not perform any actions from that state. Due to the possibility of nondeterminism after performing successively a given sequence of actions, different states can be reached. We write $p \stackrel{\epsilon}{\Longrightarrow} p'$ if $p = p'$ or there is a sequence of silent actions leading from $p$ to $p'$. A process may engage in an infinite loop of silent actions without performing any visible actions.

Let $Act$ be the set of actions of process $P$. An observer $O$ for the process $P$ is a process defined over the set of actions $Act \cup \{w\}$, where $w$ is an additional action, used by the observer to report to the external world the success of the observation. The observation is successful if during the interaction of process $P$ and observer $O$ the action $w$ is performed.

Let process $P$ and observer $O$ run concurrently and interact. We define the parallel composition $P||O$ as a process with states $(p, q) \in \mathcal{S}_P \times \mathcal{S}_O$, where $(p_0, q_0)$ is

the initial state of $P||O$ and $((p,q),\alpha,(p',q')) \in \delta_{P||O}$ iff $(p,q)$ is reachable from $(p_0,q_0)$ and either

- $(p,\tau,p') \in \delta_P$ and $q = q'$,
- $(q,\tau,q') \in \delta_O$ and $p = p'$,
- $(q,w,q') \in \delta_O$ and $p = p'$, or
- $(p,\alpha,p') \in \delta_P$ and $(q,\alpha,q') \in \delta_Q$, $\alpha \notin \{\tau, w\}$.

The process $P$ *must satisfy* the observer $O$ if in $P||O$ the action $w$ is inevitable. A process *may satisfy* the observer if in $P||O$ action $w$ is possible. For example, in Fig. 1 process $P$ may satisfy observer $O_1$, whereas it must satisfy observer $O_2$.

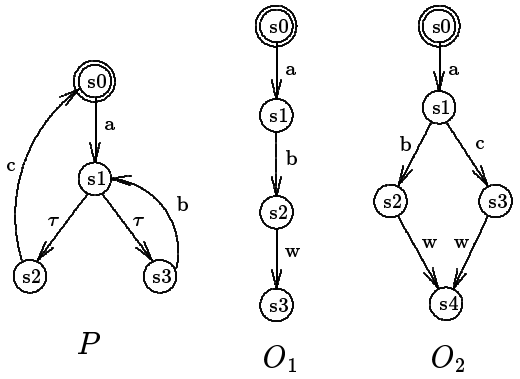


Figure 1: Process $P$ and two observers

**Definition 2.** Two processes are *must equivalent* iff they must satisfy the same sets of observers. They are *may equivalent* iff they may satisfy the same sets of observers. The processes are *testing equivalent* iff they are *must equivalent* and *may equivalent*.

## 3 Testing Equivalence via Bisimulation

A bisimulation is a notion of equivalence between processes, which is based on state matching. Intuitively, two processes are deemed equivalent if it is possible to match their states in such a way that for any two states matched, if one has an $\alpha$-derivate, then the other must have a matching $\alpha$-derivate. This can be formalized as follows.

**Definition 3.** Let $p,p' \in \mathcal{S}_P$ be states in process $P$ and $q,q' \in \mathcal{S}_Q$ states in process $Q$. Relation $R \subseteq \mathcal{S}_P \times \mathcal{S}_Q$ is a bisimulation iff $pRq$ implies the following:

1. $\forall\alpha(\,(p,\alpha,p') \in \delta_P \Rightarrow \exists q' . (q,\alpha,q') \in \delta_Q \wedge p'Rq'\,)$,
2. $\forall\alpha(\,(q,\alpha,q') \in \delta_Q \Rightarrow \exists p' . (p,\alpha,p') \in \delta_P \wedge p'Rq'\,)$.

Two processes are bisimulation equivalent iff a bisimulation $R$ matching the initial states of the processes exists. For equivalence we can set additional constraints that the matched states must satisfy.

To determine testing equivalence we have to look for a bisimulation which reflexes nondeterminism, the possibility of deadlock states, and the possibility of performing infinite sequences of silent actions in an expected way. To find such a bisimulation, we transform the processes into acceptance graphs. Let us first introduce some necessary mathematical notation.

**Definition 4.** Let $p,p' \in \mathcal{S}_P$ be states, $S$ a set of states, $\alpha \in Act$ an action, and $s \in (Act \Leftrightarrow \{\tau\})^*$ a sequence of visible actions of process $P$.

1. $\mathcal{C}_P(p) = \{\alpha \mid \exists p' \in \mathcal{S}_P \,.\, (p,\alpha,p') \in \delta_P\}$.
2. $\mathcal{D}_P(p,\alpha) = \{p' \mid (p,\alpha,p') \in \delta_P\}$,
   $\mathcal{D}_P(S,\alpha) = \bigcup_{p\in S} \mathcal{D}_P(p,\alpha)$.
3. $S^\epsilon = \{p' \mid \exists p \in S \,.\, p \stackrel{\epsilon}{\Longrightarrow} p'\}$.
4. The state $p$ of process $P$ is convergent ($p{\downarrow}\epsilon$) iff there is no infinite sequence of silent actions $\tau$ starting in $p$. The set of states $S$ is convergent ($S{\downarrow}\epsilon$) iff every state $p \in S$ is convergent.
5. Let $S$ be a set of states of process $P$. The acceptance set for $S$ and its minimal form are defined as follows:
$$\mathcal{A}_P(S) = \{\mathcal{C}_P(p) \mid p \in S \wedge \mathcal{D}_P(p,\tau) = \emptyset\}$$
$$\min \mathcal{A}_P = \{C \in \mathcal{A}_P \mid \neg\exists C' \in \mathcal{A}_P \,.\, C' \subset C\}$$

The first two definitions are straightforward. $S^\epsilon$ is the set of states which are either in the set $S$ or they are reachable from those states with a sequence of silent actions. The acceptance set $\mathcal{A}_P(S)$ represents the sets of actions which the process can perform in states $p \in S$. Note that only states incapable of silent actions are considered. Minimized acceptance set $\min \mathcal{A}_P$ contains the elements of $\mathcal{A}_P$ having no proper subsets that are also elements of $\mathcal{A}_P$. Two acceptance sets $\mathcal{A}_P$ and $\mathcal{A}_Q$ are equivalent iff $\min \mathcal{A}_P = \min \mathcal{A}_Q$.

**Definition 5.** Let $P$ be a process. Its associated acceptance graph (shortly Agraph) $P' = (\mathcal{S}_{P'}, Act, \delta_{P'}, p'_0, A_{P'})$ is an annotated deterministic process, obtained from $P$ as follows:

1. Every state $p' \in \mathcal{S}_{P'}$ is a set of states $S \subseteq 2^{\mathcal{S}_P}$ such that $S = S^\epsilon$.
2. Let $p_1,p_2 \in \mathcal{S}_{P'}$. Then $(p_1,\alpha,p_2) \in \delta_{P'}$ exactly when $\alpha \neq \tau$ and $(\mathcal{D}_P(p_1,\alpha))^\epsilon = p_2$.
3. Let $S \subseteq 2^{\mathcal{S}_P}$ be a set of states of process $P$. The state $p' = S$ is considered to be closed if $S{\downarrow}\epsilon$.
4. Let $p' \in S_{P'}$ be a set $S \subseteq 2^{\mathcal{S}_P}$. The states mapping function $A_{P'} \subseteq \mathcal{S}_{P'} \times 2^{2^{Act}} \cup \{\bot\}$ is defined as follows:
$$A_{P'}(p') = \begin{cases} \min \mathcal{A}_P(S) & \text{if } S \text{ is closed} \\ \bot & \text{otherwise} \end{cases}$$

With the states mapping function all the information about original processes, needed for determining testing equivalence, is stored in acceptance graphs. Now we are ready to express the testing equivalence as an instance of the bisimulation equivalence.

**Theorem 1.** Let $P, Q$ be processes and let $P'$ and $Q'$ be their associated Agraphs. Let $p' \in \mathcal{S}_{P'}$ and $q' \in \mathcal{S}_{Q'}$ be states in the Agraphs. The processes $P$ and $Q$ are testing equivalent iff there exists a bisimulation $R \subseteq \mathcal{S}_{P'} \times \mathcal{S}_{Q'}$ with the following two properties:

1. $(p', q') \in R$ only if $A_{P'}(p') = A_{Q'}(q')$ and
2. $(p'_0, q'_0) \in R$.

*Proof.* The proof is done in [2]. ■

Theorem 1 yields an efficient implementation of testing equivalence. After the associated Agraphs are built for the processes, testing equivalence is computed with algorithms for bisimulation equivalence.

## 4 Implementation of Testing Equivalence with BDDs

Let $P = (\mathcal{S}_P, Act, \delta_P, p_0)$ be a process and let $m = \lceil \log_2 |\mathcal{S}_P| \rceil$ and $n = \lceil \log_2 |Act| \rceil$. The states and the actions of process $P$ are encoded by vectors $\underline{r} \in \{0,1\}^m$ and $\underline{a} \in \{0,1\}^n$, respectively. The encoded states and the encoded actions are represented as minterms over variables $r_1, r_2, ...r_m$ and $a_1, a_2, ...a_n$, respectively. The set of states $S$ is represented by characteristic function $\chi_S : \{0,1\}^m \leftrightarrows \{0,1\}$ which has value 1 for all $\underline{r} \in S$ and value 0 otherwise. The set of actions is represented in the similar way. The characteristic functions are Boolean functions.

Transition relation $\delta_P$ is represented by characteristic function $\chi_{\delta_P} : \{0,1\}^m \times \{0,1\}^n \times \{0,1\}^m \leftrightarrows \{0,1\}$. To distinguish between the current and the next state in the representation of transition relation, they are encoded with the same code but represented with different variables, namely with variables $r$ and $s$, respectively. In the computer, we represent and manipulate Boolean functions efficiently by reduced ordered BDDs [1].

Let us consider the process $P$ in Fig. 2. We encode its states by vectors $s0 = (0,0,0)$, $s1 = (1,0,0)$, $s2 = (0,1,0)$, ..., $s6 = (0,1,1)$. The actions are encoded by vectors $\tau = (0,0)$, $a = (1,0)$, $b = (0,1)$, and $c = (1,1)$. The transition relation is represented by the characteristic function

$$\begin{aligned}\chi_{\delta_P} = {} & \overline{r_1}\,\overline{r_2}\,\overline{r_3}\,a_1\,\overline{a_2}\,s_1\,\overline{s_2}\,\overline{s_3} + \overline{r_1}\,\overline{r_2}\,\overline{r_3}\,a_1\,\overline{a_2}\,\overline{s_1}\,s_2\,\overline{s_3} + \\ & r_1\,\overline{r_2}\,\overline{r_3}\,\overline{a_1}\,a_2\,s_1\,s_2\,\overline{s_3} + r_1\,\overline{r_2}\,\overline{r_3}\,a_1\,a_2\,\overline{s_1}\,\overline{s_2}\,s_3 + \\ & r_1\,\overline{r_2}\,\overline{r_3}\,a_1\,\overline{a_2}\,s_1\,\overline{s_2}\,s_3 + \overline{r_1}\,r_2\,\overline{r_3}\,\overline{a_1}\,a_2\,s_1\,\overline{s_2}\,s_3 + \\ & \overline{r_1}\,r_2\,\overline{r_3}\,\overline{a_1}\,a_2\,\overline{s_1}\,s_2\,s_3 + r_1\,r_2\,\overline{r_3}\,\overline{a_1}\,\overline{a_2}\,s_1\,s_2\,\overline{s_3} + \\ & r_1\,r_2\,\overline{r_3}\,a_1\,a_2\,\overline{s_1}\,\overline{s_2}\,\overline{s_3} + r_1\,\overline{r_2}\,r_3\,a_1\,a_2\,s_1\,\overline{s_2}\,s_3 + \\ & \overline{r_1}\,r_2\,r_3\,\overline{a_1}\,\overline{a_2}\,s_1\,\overline{s_2}\,s_3.\end{aligned}$$

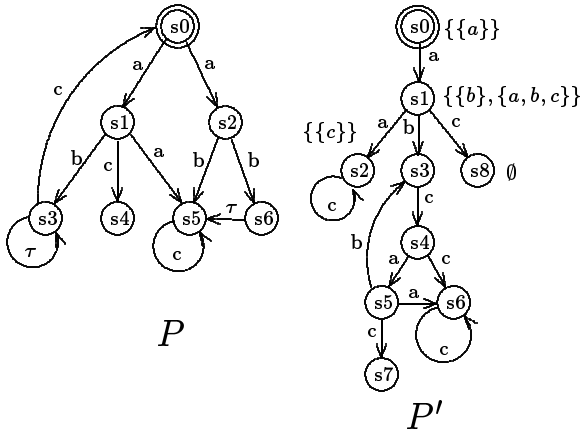


Figure 2: Process $P$ and associated Agraph $P'$. Acceptance sets for closed states $s0$, $s1$, $s2$, and $s8$ are shown beside them. Other states are not closed.

Let $S$ be a set of states of process $P$. Acceptance set $\mathcal{A}_P(S)$ is represented by Boolean function $\Psi(\mathcal{A}_P(S)) = \sum \underline{r} * \underline{a}$, where $\underline{r} \in S$, $\exists \underline{s}\,.\,(\underline{r}, \underline{a}, \underline{s}) \in \delta_P$, and $\mathcal{D}_P(\underline{r}, \tau) = \emptyset$. For example, let $S = \{s1, s2\}$ be a set of states of process $P$ in Fig. 2. Because $s1$ and $s2$ are both incapable of silent actions, their sets of actions are included in the acceptance set and we have $\mathcal{A}_P(S) = \{\{b,c,a\},\{b\}\}$. This acceptance set is represented by $\Psi(\mathcal{A}_P(S)) = r_1\,\overline{r_2}\,\overline{r_3}\,\overline{a_1}\,a_2 + r_1\,\overline{r_2}\,\overline{r_3}\,a_1\,a_2 + r_1\,\overline{r_2}\,\overline{r_3}\,a_1\,\overline{a_2} + \overline{r_1}\,r_2\,\overline{r_3}\,\overline{a_1}\,a_2$, which can be got from $\chi_{\delta_P}$ by existential quantification over variables $s$. Our implementation of testing equivalence does not require a minimization of acceptance sets. Namely, two acceptance sets $\psi_1$ and $\psi_2$, not necessary minimized, where $\psi_1$ is represented with variables $r$ and $a$, and $\psi_2$ is represented with $p$ and $a$, are equivalent, iff formula $\forall \underline{r}(\exists \underline{a}\psi_1 \Rightarrow \exists \underline{p}(\exists \underline{a}\psi_2 * \forall \underline{a}(\psi_2 \Rightarrow \psi_1))) * \forall \underline{p}(\exists \underline{a}\psi_2 \Rightarrow \exists \underline{r}(\exists \underline{a}\psi_1 * \forall \underline{a}(\psi_1 \Rightarrow \psi_2)))$ is valid.

Let the states in acceptance graph $P'$ be represented with variables $s$. We represent the acceptance sets for the whole acceptance graph by one function $\Psi(\mathcal{A}_{P'}) = \sum \underline{s} * \Psi(\mathcal{A}_P(S_{\underline{s}}))$, where $S_{\underline{s}}$ is a set of states in process $P$ corresponding to state $\underline{s} \in S_{P'}$.

A recursive function for building acceptance graphs is shown in Fig. 3. It takes process $P$, a partially constructed Agraph $P'$, a set of states $S$, a convergence flag $b$, and a partial function $\mathcal{H}$ and returns a pair consisting of Agraph and its initial state. The convergence flag $b$ denotes, whether the set of states $S$ is convergent or not.

Function AccSet creates the acceptance set for the given set of states. Functions AddNewState and AddNewTransition allocate a new state and a new transition in Agraph, respectively. Function AddNewState names allocated states with successive labels and returns them as a result. The function $\mathcal{H}(S, b)$ maps pairs comprising a set of states of the process and a convergence flag to the state in the

Agraph. It returns the label of the state in Agraph corresponding to the given set $S$ and the convergence flag $b$, if it exists. Otherwise it returns $\perp$.

BuildAG($P$, $P'$, $S$, $b$, $\mathcal{H}$) : (Agraph $\times$ State) {
  **let** $\psi =$ **if** $b$ **then** AccSet($\delta_P$, $S$) **else** $\perp$
  **in** $t :=$ AddNewState($P'$, $\psi$);
  $\mathcal{H} := \mathcal{H} \cup \{(S, b) \mapsto t\}$;
  **foreach** $\alpha \in Act \Leftrightarrow \{\tau\}, \mathcal{D}_P(S, \alpha) \neq \emptyset$ **do** {
    **let** $S_\alpha = (\mathcal{D}_P(S, \alpha))^\epsilon$, $c = (b * S_\alpha{\downarrow}\epsilon)$,
      $(P', t_{new}) =$ **if** $(\mathcal{H}(S_\alpha, c) \neq \perp)$
        **then** $(P', \mathcal{H}(S_\alpha, c))$
        **else** BuildAG($P$, $P'$, $S_\alpha$, $c$, $\mathcal{H}$)
    **in** AddNewTransition($P'$, $(t, \alpha, t_{new})$);
  }
  **return** $(P', t)$;
}

Figure 3: The function for building Agraphs

The function for testing equivalence (Fig. 4) takes processes $M$ and $N$ and returns a boolean. First, Agraphs for both processes are constructed. Then the pairs of compatible states in Agraphs are determined in the following three steps. The non-closed states are compatible, the states with empty acceptance set are compatible, and the states with equivalent non-empty acceptance sets are also compatible.

Function Bisimulation takes a pair of initial states, an equivalence relation and two transition relations. It computes the greatest fixed point of the function defining the equivalence relation and returns true iff it contains the pair of initial states. The initial equivalence relation is the sum of those pairs of compatible states that can perform the same actions [6].

TestingEquivalence ($M$, $N$) : Boolean {
  **let** $S_M = \{m_0\}^\epsilon$, $M_0' = (\emptyset, Act, \emptyset, \perp, \emptyset)$,
    $S_N = \{n_0\}^\epsilon$, $N_0' = (\emptyset, Act, \emptyset, \perp, \emptyset)$;
  **in** {
    $(M', m_0') :=$ BuildAG($M$, $M_0'$, $S_M$, $S_M{\downarrow}\epsilon$, $\emptyset$);
    $(N', n_0') :=$ BuildAG($N$, $N_0'$, $S_N$, $S_N{\downarrow}\epsilon$, $\emptyset$); }
  $C_\perp := 0$; $C_\emptyset := 0$;
  **foreach** $(m, n)$, $m \in \mathcal{S}_{M'}$, $\mathcal{A}_{M'}(m) = \perp$,
    $n \in \mathcal{S}_{N'}$, $\mathcal{A}_{N'}(n) = \perp$ **do** $C_\perp := C_\perp + m * n_{r \to p}$;
  **foreach** $(m, n)$, $m \in \mathcal{S}_{M'}$, $\mathcal{A}_{M'}(m) = \emptyset$,
    $n \in \mathcal{S}_{N'}$, $\mathcal{A}_{N'}(n) = \emptyset$ **do** $C_\emptyset := C_\emptyset + m * n_{r \to p}$;
  **let** $\Psi_M = \Psi(\mathcal{A}_{M'})$, $\Psi_N = \Psi(\mathcal{A}_{N'})_{r \to p, s \to q}$
  **in** $C := \forall \underline{r}(\exists \underline{a} \Psi_M \Rightarrow \exists \underline{p}(\exists \underline{a} \Psi_N * \forall \underline{a}(\Psi_N \Rightarrow \Psi_M))) *$
    $\forall \underline{p}(\exists \underline{a} \Psi_N \Rightarrow \exists \underline{r}(\exists \underline{a} \Psi_M * \forall \underline{a}(\Psi_M \Rightarrow \Psi_N)))$;
  **let** $\chi_{Act} := \forall \underline{a}(\exists \underline{s} \delta_M \Leftrightarrow (\exists \underline{s} \delta_N)_{r \to p})$,
    $\chi_{Eq} := (C_\perp + C_\emptyset + C_{s \to r, q \to p}) * \chi_{Act}$
  **in** $r :=$ Bisimulation($(m_0', n_0')$, $\chi_{Eq}$, $\delta_{M'}$, $\delta_{N'}$);
  **return** $r$;
}

Figure 4: The function for testing equivalence

## 5 Conclusion

This paper presents an efficient implementation of a testing equivalence checking. The observed systems are described as processes and represented with BDDs. The algorithm for testing equivalence is composed of transforming processes into Agraphs and then determining appropriate bisimulation equivalence between them. Although the problem of computing testing equivalence is known to be PSPACE-complete, in practice for many systems it can be computed in acceptable time. The efficiency of the computation depends on the implementation of the BDD package used [1].

Testing equivalence (known also as failure equivalence) is conceptually simple and close to the concept of testing. It is not equal to bisimulation equivalence. Some processes are testing equivalent, but not bisimulation equivalent, and vice versa. Some variations of the testing equivalence have been defined, which treated cyclic behaviours and infinite computations (i.e. divergencies) more or less strictly [4, 5, 2].

A similar approach to equivalence checking is preorder checking [3]. The algorithm for determining testing equivalence can be easily extended to compute testing preorder. Other possible extensions of presented work are system abstraction and diagnostics generation. The idea of abstraction is in finding the minimal system testing equivalent to the given one. Diagnostic features are used to explain why two systems are not testing equivalent.

## References


[1] A. Časar and R. Meolic. Representation of Boolean functions with ROBDDs, 1993. To be published in IEEE Student Papers Book.

[2] R. Cleaveland and M. Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, pages 1–20, 1993.

[3] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[4] F. Corno, M. Cusinato, M. Ferrero, and P. Prinetto. Proving Testing Preorders for Process Algebra Descriptions. Politecnico di Torino, Italy.

[5] D. Larrabeiti, J. Quemada, and S. Pavon. A Practical Approach to Testing Finite State Systems. In *Proceedings of $2^{nd}$ International Workshop on Applied Formal Methods in System Design*, pages 17–24, Zagreb, Croatia, 1997.

[6] M. Sepešy, T. Kapus, and B. Horvat. Verification of systems communication behaviour using binary decision diagrams. In *Proceedings of the Fifth Electrotechnical and Computer Science Conference ERK'96, Portorož, Slovenia*, volume B, pages 11–14, 1996. (in Slovene).