A Method of a Proof of Observational Equivalence of Processes

Andrew M. Mironov

The Institute of Informatics Problems of the Russian Academy of Sciences
amironov66@gmail.com

Abstract. In the paper we introduce a concept of a process with message passing and present a method of checking observational equivalence of processes with message passing. The presented method is illustrated by an example of a verification of sliding window protocol.

Keywords: processes, message passing, verification, sliding window protocol

1 Introduction

The problem of formal representation and verification of discrete processes is one of the most important problems in computer science. There are several approaches to this problem, the main of them are: CCS and π-calculus [1], [2], CSP and its generalizations [3], temporal logic and model checking [4], Petri nets [5], process algebras [6], communicating finite-state machines [7].

In the present paper we introduce a new model of discrete processes, which is a synthesis of Milner's model of processes [1] and the model of communicating finite-state machines [7]. Discrete processes are represented in our model as graphs, edges of which are labelled by operators. These operators consist of internal actions and communication actions. Proofs of correctness of processes are represented by sets of formulas, associated with pairs of states of analyzed processes. This method of verification of processes is a synthesis of Milner's approach related on the concept of an observational equivalence [1] and Floyd's inductive assertion method [8]. For a simplification of an analysis of processes we introduce a simplification operation on processes. With use this operation it is possible to reduce a complexity of verification of processes. We illustrate an advantage of the proposed model and the verification method on the example of verification of a two-way sliding window protocol.

2 Motivation, advantages of the proposed approach and its comparison with other works

2.1 Motivation of the proposed approach

The main disadvantage of modern methods of verification of discrete processe is their large complexity. More precisely,

- the main disadvantage of verification methods based on model checking approach is a high computational complexity related to the state explosion problem, and
- disadvantages of methods based on theorem proving approach are related with a high complexity of construction of corresponging theorems and their proofs, and also with an understanding of these proofs.

For example, in recent paper [9] a complete presentation of proofs of theorems related to verification of two-way sliding window protocol takes a few dozen pages of a complex mathematical text.

The main motivation for the proposed approach to modeling and verification of discrete systems by checking of observational equivalence of corresponded processes with message passing is to simplify and make more obvious the following aspects of modeling and analysis of discrete systems: representation of mathematical models of analyzed systems, construction of proofs of correctness of the systems, and understanding of these proofs by any who is not a strong expert in the mathematical theory of verification of discrete systems.

2.2 Advantages of the proposed approach

The proposed mathematical model of processes with message passing allows to construct such mathematical models of analysed systems that are very similar to an original description of these systems on any imperative programming language. In section 10 we give an example of such model that corresponds to a C-program describing a sliding window protocol using go back n (the program was taken from book [10], section 3.4.2).

The main advantage of the proposed approach is a possibility to use a simplification operation of models of analyzed systems, that allows essentially simplify the problem of verification of these models. In section 10 we present a result of such simplification for the above model of a sliding window protocol: this model can be simplified to a model with only one state. It should be noted also that the simplified models allow more clearly understand main features of analyzed systems, and facilitate a construction of correctness proofs for analyzed systems.

If an analyzed property of a system has the form of a behavior which is described by some process, for example, in the case when

- an analyzed system is a network protocol, and
- a property of this system is a description of an external behavior of this protocol (related to its interaction with a higher-level protocol)

then a proof of a correctness of such system in this model is a set of formulas associated with pairs of states, the first of which is a state of the analyzed system, and the second is a state of a a process which describes a property of the analyzed system.

In section 10 we give an example of such proof, which is a small set of simple formulas. These formulas can be naturally derived from a simplified model of an analyzed protocol.

Another advantage of the proposed approach is a possibility to verify systems with unbounded sets of states. One of examples of such systems is the above sliding window protocol using go back n.

2.3 Comparison with other works

In this section we present an overview of papers related to verification of message passing systems, which are most relevant to the present paper.

The paper [9] deals with modeling and manual verification in the process algebraic language μCRL. Authors use the theorem prover PVS to formalize and to mechanically prove the correctness of a protocol using selective repeat (a C-program describing this protocol is presented in section 3.4.3 of the book [10]). The main disadvantage of this work is a large complexity of proofs of theorems related to verification of this protocol. This protocol can be verified more simply with use of the approach proposed in the present paper.

There are a lot of works related to verification of systems with message passing based on temporal logic and model checking approach. Most relevant ones to the present paper are [11], [12], [13], [14], [15], [16], [17]. The most deficiency of all of them is restricted abilities: these methods allow verify only finite state systems.

Among other approaches it should be noted approaches with use of first order logic and assertional verification: [18], [19], and approaches with use of process algebra: [20], [21], [22], [23]. The most deficiency of these approaches is a high complexity of construction of proofs of correctness of analyzed systems.

3 Auxiliary concepts

3.1 Terms

We assume that there are given a set $\mathcal{X}$ of **variables**, a set $\mathcal{D}$ of **values**, a set $\mathcal{C}$ of **constants**, and a set $\mathcal{F}$ of **function symbols**. Any constant from $\mathcal{C}$ is interpreted by a value from $\mathcal{D}$, and any function symbol from $\mathcal{F}$ is interpreted by an operation on $\mathcal{D}$.

We assume that $\mathcal{C}$ contains constants 0 and 1, and $\mathcal{F}$ contains boolean function symbols $\wedge, \vee, \to$, which correspond to standard boolean operations on $\{0, 1\}$.

The set $\mathcal{E}$ of **terms** is defined in the standard way. Variables and constants are terms. Other terms have the form $f(e_1, \ldots, e_n)$, where $f \in \mathcal{F}$, and $e_1, \ldots, e_n$ are terms. For each $e \in \mathcal{E}$ a set of all variables occurring in e is denoted by X_e.

If $X \subseteq \mathcal{X}$, then a **valuation** of variables of X is a correspondence ξ, that associates each variable $x \in X$ with a value $x^\xi \in \mathcal{D}$. We denote by the record $X^\bullet$ the set of all valuations of variables from X. For each $e \in \mathcal{E}$, each $X \supseteq X_e$ and each $\xi \in X^\bullet$ the record e^ξ denotes an object called a **value** of e on ξ and defined in the standard way. We assume that terms e_1 and e_2 are equal iff $\forall \xi \in (X_{e_1} \cup X_{e_2})^\bullet \; e_1^\xi = e_2^\xi$.

A term e is a **formula** if $\forall \xi \in X_e^\bullet$ the value e^ξ is 0 or 1. The set of all formulas is denoted by $\mathcal{B}$. The symbols $\top$ and $\bot$ denote true and false formula respectively. We shall write formulas of the form $\wedge(b_1, b_2)$, $\vee(b_1, b_2)$, etc. in a more familiar form $b_1 \wedge b_2$, $b_1 \vee b_2$, etc.

3.2 Atomic operators

We assume that there is given a set $\mathcal{N}$, whose elements are considered as names of objects that can be sent or received by processes.

An **atomic operator (AO)** is an object o of one of three forms presented below. Each pair (o, ξ), where o is an AO, and ξ is a valuation of variables occurred in o, corresponds to an action o^ξ, informally defined below.

1. An **input** is an AO of the form $\alpha?x$, where $\alpha \in \mathcal{N}$ and $x \in \mathcal{X}$. An action $(\alpha?x)^\xi$ is a receiving from another process an object named α, with a message attached to this object, this message is assigned to the variable x.
2. An **output** is an AO of the form $\alpha!e$, where $\alpha \in \mathcal{N}$ and $e \in \mathcal{E}$. An action $(\alpha!e)^\xi$ is a sending to another process an object named α, to which a message e^ξ is attached.
3. An **assignment** is an AO of the form $x := e$, where $x \in \mathcal{X}$, $e \in \mathcal{E}$. An action $(x := e)^\xi$ is an assigning the variable x with the value e^ξ.

Below we use the following notations.

- For each AO o the record X_o denotes the set of all variables occurred in o.
- If $e \in \mathcal{E}$, and o is an assignment, then the record $o(e)$ denotes a term defined as follows: let o has the form $(x := e')$, then $o(e)$ is obtained from e by a replacement of all occurrences of the variable x by the term e'.
- If o is an assignment, and $\xi \in X^\bullet$, where $X_o \subseteq X \subseteq \mathcal{X}$, then the record $\xi \cdot o$ denotes a valuation from $X^\bullet$, defined as follows: let $o = (x := e)$, then $x^{\xi \cdot o} = e^\xi$ and $\forall\, y \in X \setminus \{x\}\;\; y^{\xi \cdot o} = y^\xi$.

It is easy to prove that if o is an assignment and $e \in \mathcal{E}$, then for each $\xi \in X^\bullet$, where $X_o \cup X_e \subseteq X \subseteq \mathcal{X}$, the equality $o(e)^\xi = e^{\xi \cdot o}$ holds. This equality is proved by an induction on the structure of the term e.

3.3 Operators

An **operator** is a record O of the form $b\,[o_1, \ldots, o_n]$, where b is a formula called a **precondition** of O (this formula will be denoted as $\langle O \rangle$), and $o_1, \ldots, o_n$ is a sequence of AOs (this sequence will be denoted as $[O]$), among which there is at most one input or output. The sequence $[O]$ may be empty ($[\,]$).

If $[O]$ contains an input (or an output) then O is called an **input operator** (or an **output operator**), and in this case the record N_O denotes a name occurred in O. If $[O]$ does not contain inputs and outputs, then we call O an **internal operator**.

If $\langle O \rangle = \top$, then such precondition can be omitted in a notation of O.

Below we use the following notations.

1. For each operator O a set of all variables occurred in O is denoted by X_O.

2. If O is an operator, and $b \in \mathcal{B}$, then the record $O \cdot b$ denotes an object, which either is a formula or is not defined. This object is defined recursively as follows. If $[O]$ empty, then $O \cdot b \stackrel{\text{def}}{=} \langle O \rangle \wedge b$. If $[O] = o_1, \ldots, o_n$, where $n \geq 1$, then we shall denote by the record $O \setminus o_n$ an operator obtained from O by a removing of its last AO, and
   - if $o_n = \alpha?x$, then $O \cdot b \stackrel{\text{def}}{=} (O \setminus o_n) \cdot b$, if $x \notin X_b$, and is undefined otherwise
   - if $o_n = \alpha!e$, then $O \cdot b \stackrel{\text{def}}{=} (O \setminus o_n) \cdot b$
   - if $o_n = (x := e)$, then $O \cdot b \stackrel{\text{def}}{=} (O \setminus o_n) \cdot o_n(b)$.
3. If O is an internal operator, and $\xi \in X^\bullet$, where $X_O \subseteq X \subseteq \mathcal{X}$, then the record $\xi \cdot O$ denotes a valuation from $X^\bullet$, defined as follows: if $[O]$ is empty, then $\xi \cdot O \stackrel{\text{def}}{=} \xi$, and if $[O] = o_1, \ldots, o_n$, where $n \geq 1$, then $\xi \cdot O \stackrel{\text{def}}{=} (\xi \cdot (O \setminus o_n)) \cdot o_n$.

It is easy to prove that if O is internal and $b \in \mathcal{B}$, then for each $\xi \in X^\bullet$, where $X_O \cup X_b \subseteq X \subseteq \mathcal{X}$, such that $\langle O \rangle^\xi = 1$, the equality $(O \cdot b)^\xi = b^{\xi \cdot O}$ holds. This equality is proved by an induction on a lenght of $[O]$.

3.4 Concatenation of operators

Let O_1 and O_2 be operators, and at least one of them is internal.

A **concatenation** of O_1 and O_2 is an object denoted by the record $O_1 \cdot O_2$, that either is operator or is undefined. This object is defined iff $O_1 \cdot \langle O_2 \rangle$ is defined, and in this case $O_1 \cdot O_2 \stackrel{\text{def}}{=} (O_1 \cdot \langle O_2 \rangle)[[O_1], [O_2]]$. It is easy to prove that

- if operators O_1, O_2 and formula b are such that objects in both sides of the equality $(O_1 \cdot O_2) \cdot b = O_1 \cdot (O_2 \cdot b)$ are defined, then this equality holds, and
- if operators O_1, O_2, O_3 are such that all objects in both sides of the equality $(O_1 \cdot O_2) \cdot O_3 = O_1 \cdot (O_2 \cdot O_3)$ are defined, then this equality holds.

4 Processes with a message passing

4.1 A concept of a process with a message passing

A **process with a message passing** (also called more briefly a **process**) is a 4-tuple P of the form

$$P = (S_P, s_P^0, T_P, I_P) \tag{1}$$

components of which have the following meanings.

- S_P is a set of **states** of the process P.
- $s_P^0 \in S_P$ is an **initial state** of the process P.
- T_P is a set of **transitions** of the process P, each transition from T_P has the form $s_1 \stackrel{O}{\to} s_2$, where $s_1, s_2 \in S_P$ and O is an operator.
- $I_P \in \mathcal{B} \setminus \{\bot\}$ is a **precondition** of the process P.

A transition $s_1 \overset{O}{\to} s_2$ is called an **input**, an **output**, or an **internal** transition, if O is an input operator, an output operator, or an internal operator, respectively.

For each process P

- the record X_P denotes the set consisting of
  - all variables occurred in any of the transitions from T_P, or in I_P, and
  - a variable at_P, which is not occurred in I_P, and in transitions from T_P, the set of values of at_P is S_P
- the record $\langle P\rangle$ denotes the formula $(at_P = s^0_P) \wedge I_P$.

For each transition $t \in T_P$ the records O_t, $\langle t\rangle$, $start(t)$ and $end(t)$ denote an operator, a formula and states defined as follows: if t has the from $s_1 \overset{O}{\to} s_2$, then

$$O_t \overset{\text{def}}{=} O, \quad \langle t\rangle \overset{\text{def}}{=} (at_P = s_1) \wedge \langle O\rangle, \quad start(t) \overset{\text{def}}{=} s_1, \quad end(t) \overset{\text{def}}{=} s_2.$$

If t is an input or an output, then the record N_t denotes the name N_{O_t}.

A set X^s_P of **essential variables** of P is a smallest (w.r.t. inclusion) set satisfying the following conditions.

- X^s_P contains all variables contained in preconditions and outputs in operators O_t, where $t \in T_P$.
- If P contains an AO $x := e$ and $x \in X^s_P$, then X^s_P contains all variables occurred in e.

A process P is associated with a graph denoted by the same symbol P. Vertices of this graph are states of P, and its edges correspond to transitions of P: each transition $s_1 \overset{O}{\to} s_2$ corresponds to an edge from s_1 to s_2 with a label O.

4.2 Actions of processes

An **action of a process** (or, briefly, an **action**) is a record of one of the following three forms.

- $\alpha?d$, where $\alpha \in \mathcal{N}$ and $d \in \mathcal{D}$. An action of this form is called a **receiving** of an object named α with the attached message d.
- $\alpha!d$, where $\alpha \in \mathcal{N}$ and $d \in \mathcal{D}$. An action of this form is called a **sending** of an object named α with the attached message d.
- τ. An action of this form is called a **silent action**.

A set of all actions is denoted by $\mathcal{A}$.

4.3 An execution of a process

An **execution** of a process (1) is a walk on the graph P starting from s^0_P, with an execution of AOs occurred in labels of traversed edges. At each step $i \geq 0$ of this walk there is defined a current state $s_i \in S_P$ and a current valuation $\xi_i \in X^\bullet_P$. We assume that $s_0 = s^0_P$, $\langle P\rangle^{\xi_0} = 1$, and for each step i of this walk $at^{\xi_i}_P = s_i$.

An execution of P on step i is described informally as follows. If there is no transitions in T_P starting at s_i, then P terminates, otherwise

- P selects a transition $t \in T_P$, such that $\langle t\rangle^{\xi_i} = 1$, and if t is an input or an output, then at the current moment P can receive or send respectively an object named N_t (i.e. at the same moment there is another process that can send to P or receive from P respectively an object named N_t). If there is no such transition, then P suspends until at least one such transition will appear, and after resumption its execution P selects one of such transitions,
- after a sequential execution of all AOs occurred in the operator O_t of the selected transition t, P moves to the state $end(t)$.

An execution of each AO o occurred in $[O_t]$ consists of a performing of an action $a \in \mathcal{A}$ and a replacement the current valuation ξ on a valuation ξ', which is considered as a current valuation after an execution of the AO o. An execution of an AO o is as follows:

- if $o = \alpha?x$, then P performs an action of the form $\alpha?d$, and $x^{\xi'} \stackrel{\text{def}}{=} d$, $\forall\, y \in X_P \setminus \{x\} \quad y^{\xi'} \stackrel{\text{def}}{=} y^{\xi}$
- if $o = \alpha!e$, then P performs the action $\alpha!(e^{\xi})$, and $\xi' \stackrel{\text{def}}{=} \xi$
- if $o = (x := e)$, then P performs τ, and $x^{\xi'} \stackrel{\text{def}}{=} e^{\xi}$, $\forall\, y \in X_P \setminus \{x\} \quad y^{\xi'} \stackrel{\text{def}}{=} y^{\xi}$.

5 Operations on processes

In this section we define some operations on processes which can be used for a construction of complex processes from simpler ones. These operations are generalizations of corresponded operations on processes defined in Milners's Calculus of Communicating Systems [1].

5.1 Parallel composition

The operation of parallel composition is used for building processes, composed of several communicating subprocesses.

Let $P_i = (S_i, s^0_i, T_i, I_i) \quad (i = 1, 2)$ be processes, such that $S_1 \cap S_2 = \emptyset$ and $X_{P_1} \cap X_{P_2} = \emptyset$. A **parallel composition** of P_1 and P_2 is a process $P = (S_, s^0, T, I)$, where

$$S \stackrel{\text{def}}{=} S_1 \times S_2, \quad s^0 \stackrel{\text{def}}{=} (s^0_1, s^0_2), \quad I \stackrel{\text{def}}{=} I_1 \wedge I_2$$

and T consists of the following transitions:

- for each transition $s_1 \overset{O}{\to} s_1'$ of the process P_1, and each state s of P_2 the process P has the transition $(s_1, s) \overset{O}{\to} (s_1', s)$
- for each transition $s_2 \overset{O}{\to} s_2'$ of the process P_2, and each state s of the process P_1 the process P has the transition $(s, s_2) \overset{O}{\to} (s, s_2')$
- for each pair of transition of the form $\begin{cases} s_1 \overset{O_1}{\to} s_1' \in T_{P_1} \\ s_2 \overset{O_2}{\to} s_2' \in T_{P_2} \end{cases}$ where one of the operators O_1, O_2 has the form $(O_1' \cdot [\alpha?x]) \cdot O_1''$, and another operator has the form $(O_2' \cdot [\alpha!e]) \cdot O_2''$, the process P has the transition $(s_1, s_2) \overset{O}{\to} (s_1', s_2')$, where $\langle O \rangle = \langle O_1 \rangle \wedge \langle O_2 \rangle$ and $[O] = \Big((O_1' \cdot O_2') \cdot [x := e]\Big) \cdot (O_1'' \cdot O_2'')$.

A parallel composition of P_1 and P_2 is denoted by the record $P_1 \,|\, P_2$.

If $S_1 \cap S_2 \neq \emptyset$ or $X_{P_1} \cap X_{P_2} \neq \emptyset$, then before a construction of the process $P_1 \,|\, P_2$ it is necessary to replace states and variables occuring in both processes on new states or variables respectively.

For any tuple $P_1, P_2, \ldots, P_n$ of processes their parallel composition $P_1 \,|\, \ldots \,|\, P_n$ is defined as the process $((P_1 \,|\, P_2) \,|\, \ldots) \,|\, P_n$.

5.2 Restriction

Let $P = (S, s^0, T, I)$ be a process, and L be a subset of the set $\mathbf{N}$.

A **restriction** of P with respect to L is the process $P \setminus L = (S, s^0, T', I)$ which is obtained from P by removing of those transitions that have labels with the names from L, i.e.

$$T' \overset{\text{def}}{=} \big\{ (\, s \xrightarrow{\quad O \quad} s' \,) \in R \,\big|\, [O] = [\,], \ \text{ or } N_O \not\in L \big\}$$

6 Renaming

The last operation is called a **renaming**: for any mapping $f : \mathcal{N} \to \mathcal{N}$ and any process P the record $P[f]$ denotes a process which is called a renaming of P and is obtained from P by changing of names occurred in P: any name α occurred in P is changed on $f(\alpha)$.

If the mapping f acts non-identically only on the names $\alpha_1, \ldots, \alpha_n$, and maps them to the names $\beta_1, \ldots, \beta_n$ respectively, then the process $P[f]$ can be denoted also as $P[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$.

6.1 An example of a process defined with use of parallel composition and restriction

In this subsection we describe a process which is defined with use of the above operations. This process is an implementation of a distributed algorithm of separation of sets. The problem of separation of sets has the following form. Let U, V be a pair of finite disjoint sets, with each element $x \in U \cup V$ is associated with a number $weight(x)$, called a **weight** of this element. It is need to convert this pair to a pair of sets U', V', such that

- $|U| = |U'|, \quad |V| = |V'|$
  (for each finite set M the notation $|M|$ denotes a number of elements in M)
- $\forall\, u \in U', \forall\, v \in V' \quad weight(u) \leq weight(v)$.

Below we shall call U and V as the left set and the right set, respectively.

The problem of separation of sets can be solved by an execution of several sessions of exchange elements between these sets. Each session consists of the following actions:

- find an element mx with a maximum weight in the left set
- find an element mn with minimum weight in the right set
- transfer
  - mx from the left set to the right set, and
  - mn from the right set to the left set.

To implement this idea it is proposed a distributed algorithm, defined as a process of the form

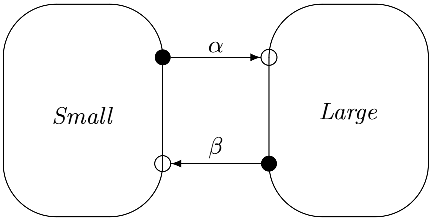
$$(Small \mid Large) \setminus \{\alpha, \beta\} \tag{2}$$

where

- the process *Small* executes operations associated with the left set, and
- the process *Large* executes operations associated with the right set.

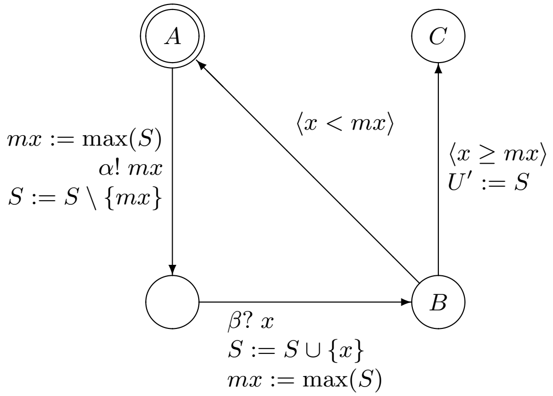
The restriction of the actions with names α and β in (2) means that a transmission of objects with names α and β can be executed only between the subprocesses *Small* and *Large*, i.e. such objects can not be transmitted outside the process (2).

A flow graph (i.e. a relation between components) corresponded to this process has the form



Below we shall use the following notations: for each subset $W \subseteq U \cup V$ the records $\max(W)$ and $\min(W)$ denote an element of W with maximum and minimum weight, respectively. A similar meaning have the records $\max(W)$ and $\min(W)$, where W is a a variable whose values are subsets of $U \cup V$.

The process *Small* has the following form:

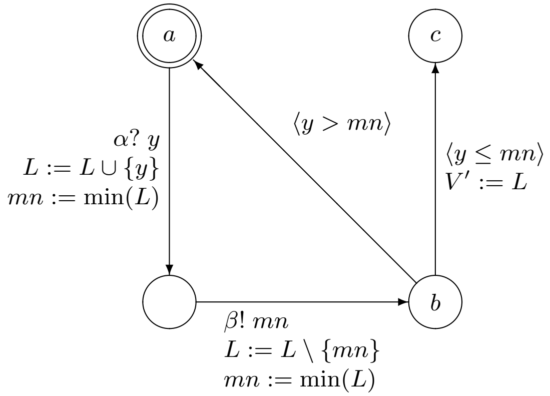


(3)

(a double circle denotes an initial state).

An initial condition of the process *Small* is $(S = U)$.

The process *Large* has the following form:



(4)

An initial condition of the process *Large* is $(L = V)$.

7 Realizations of processes

7.1 Realizations of AOs and sequences of AOs

A **realization of an AO** o is a triple (ξ, a, ξ'), such that

- $\xi, \xi' \in X^{\bullet}$, where $X_o \subseteq X \subseteq \mathcal{X}$, and $a \in \mathcal{A}$
- if $o = \alpha?x$, then $a = \alpha?(x^{\xi'})$ and $\forall\, y \in X \setminus \{x\} \quad y^{\xi'} = y^{\xi}$
- if $o = \alpha!e$, then $a = \alpha!(e^{\xi})$ and $\xi' = \xi$
- if $o = (x := e)$, then $a = \tau$ and $\xi' = \xi \cdot o$.

Let $o_1, \ldots, o_n$ be a sequence of AOs which contains at most one input or output. A **realization of** $o_1, \ldots, o_n$ is a triple (ξ, a, ξ'), such that

- $\xi, \xi' \in X^{\bullet}$, where $X \subseteq \mathcal{X}$ and $a \in \mathcal{A}$

– if $n = 0$, then $\xi' = \xi$ and $a = \tau$, otherwise there exists a sequence

$$(\xi_0, a_1, \xi_1),\ (\xi_1, a_2, \xi_2),\ \ldots, (\xi_{n-1}, a_n, \xi_n) \tag{5}$$

where $\xi_0 = \xi$, $\xi_n = \xi'$, $\forall\, i = 1, \ldots, n$ (ξ_{i-1}, a_i, ξ_i) is a realization of o_i, and $a = \tau$, if each a_i in (5) is equal to τ, otherwise a coincides with that a_i, which is different from τ.

7.2 Realization of transitions

Let P be a process of the form (1), and $t \in T_P$.

A **realization of** t is a triple (ξ_1, a, ξ_2), where $\xi_1, \xi_2 \in X_P^\bullet$ and $a \in \mathcal{A}$, such that $\langle t\rangle^{\xi_1} = 1$ and $(\xi_1 \cdot (at_P := end(t)), a, \xi_2)$ is a realization of $[O_t]$.

The following properties hold.

– If a transition t is internal or is an output, then for each $\xi \in X_P^\bullet$, such that $\langle t\rangle^\xi = 1$, there exist a unique $\xi' \in X_P^\bullet$ and a unique $a \in \mathcal{A}$, such that (ξ, a, ξ') is a realization of t. We shall denote such ξ' by $\xi \cdot t$.

– If a transition t is an input, then for each $\xi \in X_P^\bullet$, such that $\langle t\rangle^\xi = 1$, and each $d \in \mathcal{D}$ there exists a unique $\xi' \in X_P^\bullet$, such that $(\xi, N_t?d, \xi')$ is a realization of t. We shall denote such ξ' by $\xi \cdot t^d$.

7.3 Realizations of processes

A **realization of a process** P is a graph P^r having the following components.

– The set S_P^r of vertices of P^r is the disjoint union $X_P^\bullet \cup \{P^0\}$.

– The set T_P^r of edges of P^r consists of the following edges:
  - for each realization (ξ_1, a, ξ_2) of any $t \in T_P$ the graph P^r has an edge from ξ_1 to ξ_2 with a label a, and
  - for each $\xi \in X_P^\bullet$, such that $\langle P\rangle^\xi = 1$, and each edge of P^r from ξ to ξ' with a label a the graph P^r has an edge from P^0 to ξ' with a label a.

We shall use the following notations: for any pair v, v' of vertices of P^r

– the record $v_1 \xrightarrow{a} v_2$ denotes an edge from v_1 to v_2 with a label a

– $v \xrightarrow{\tau^*} v'$ means that either $v = v'$ or $\exists\, v_0, v_1, \ldots, v_n : \forall\, i = 1, \ldots, n$ the graph P^r has an edge $v_{i-1} \xrightarrow{\tau} v_i$, and $v_0 = v$, $v_n = v'$.

– $v \xrightarrow{\tau^* a \tau^*} v'$ (where $a \in \mathcal{A}$) means that $\exists\, v_1, v_2$: the graph P^r has an edge $v_1 \xrightarrow{a} v_2$, and $v \xrightarrow{\tau^*} v_1$, $v_2 \xrightarrow{\tau^*} v'$.

8 Observational equivalence of processes

8.1 A concept of observational equivalence of processes

Processes P_1 and P_2 are said to be **observationally equivalent** if P_1^r and P_2^r are observationally equivalent in Milner's sense [1], i.e. there exists $\mu \subseteq S_{P_1}^r \times S_{P_2}^r$, such that

1. $(P_1^0, P_2^0) \in \mu$
2. if $(v_1, v_2) \in \mu$ and $v_1 \xrightarrow{\tau} v_1'$, then $\exists\, v_2' : v_2 \xrightarrow{\tau^*} v_2',\ (v_1', v_2') \in \mu$,
   if $(v_1, v_2) \in \mu$ and $v_2 \xrightarrow{\tau} v_2'$, then $\exists\, v_1' : v_1 \xrightarrow{\tau^*} v_1',\ (v_1', v_2') \in \mu$
3. if $(v_1, v_2) \in \mu$ and $v_1 \xrightarrow{a} v_1'$, where $a \neq \tau$, then $\exists\, v_2' : v_2 \xrightarrow{\tau^* a \tau^*} v_2',\ (v_1', v_2') \in \mu$,
   if $(v_1, v_2) \in \mu$ and $v_2 \xrightarrow{a} v_2'$, where $a \neq \tau$, then $\exists\, v_1' : v_1 \xrightarrow{\tau^* a \tau^*} v_1',\ (v_1', v_2') \in \mu$

The record $P_1 \approx P_2$ means that P_1 and P_2 are observationally equivalent.

A lot of problems related to verification of discrete systems can be reduced to the problem to prove that $P_1 \approx P_2$, where the process P_1 is a model of a system being analyzed, and P_2 is a model of some property of this system. In section 10 we consider an example of a proof that $P_1 \approx P_2$, where P_1 is a model of the sliding window protocol, and P_2 is a model of its external behavior.

8.2 A method of a proof of observational equivalence of processes

In this section we present a method of a proof of observational equivalence of processes. This method is based on theorem 1. To formulate and prove this theorem, we introduce auxiliary concepts and notations.

1. Let P be a process, and $s, s' \in S_P$. A **composite transition (CT)** from s to s' is a sequence T of transitions of P of the form
   $$s = s_0 \xrightarrow{O_1} s_1, \quad s_1 \xrightarrow{O_2} s_2, \quad \ldots \quad s_{n-1} \xrightarrow{O_n} s_n = s' \tag{6}$$
   such that there is at most one input or output operator among $O_1, \ldots, O_n$, and there are defined all concatenations in the expression
   $$(\ldots(O_1 \cdot O_2) \cdot \ldots) \cdot O_n \tag{7}$$
   Sequence (6) may be empty, in this case $s = s'$. If CT T is not empty and has the form (6), then the record O_T denotes a value of the expression (7). If CT T is empty, then $O_T \overset{\text{def}}{=} [\,]$.
   We shall use for CTs the same concepts and notation as for ordinary transitions ($start(T)$, $end(T)$, N_T etc.). A CT T is said to be an input, an output, or an internal iff O_T is an input operator, an output operator, or an internal operator, respectively.
   A concept of a realization of a CT is defined by analogy with the concept of a realization of a transition (see section 7.2). This concept has properties similar to properties of a realization of a transition, in particular:
   (a) if a CT T is internal or is an output, then for each $\xi \in X_P^\bullet$, such that $\langle T\rangle^\xi = 1$, there is a unique $\xi' \in X_P^\bullet$ and a unique $a \in \mathcal{A}$, such that (ξ, a, ξ') is a realization of T, we shall denote such ξ' by the record $\xi \cdot T$
   (b) if a CT T is an input, then for each $\xi \in X_P^\bullet$, such that $\langle T\rangle^\xi = 1$, and each $d \in \mathcal{D}$ there is a unique $\xi' \in X_P^\bullet$, such that $(\xi, N_T?d, \xi')$ is a realization of T, we shall denote such ξ' by the record $\xi \cdot T^d$.

2. If b and b' are formulas, then the record $b \leq b'$ is a brief notation of the proposition that the formula $b \to b'$ is true.
3. If O_1, O_2 are operators, and $b \in \mathcal{B}$, then the record $(O_1, O_2) \cdot b$ denotes a formula defined by a recursive definition presented below. In this definition we use records of the form $O \setminus o$ and $o(b)$, which denote an operator and a formula respectively, defined in section 3.3.
   Let $[O_1] = o_1, \ldots, o_n$ and $[O_2] = o'_1, \ldots, o'_m$, then the formula
$$(O_1, O_2) \cdot b \tag{8}$$
   is defined as follows:
   (a) $\langle O_1 \rangle \wedge \langle O_2 \rangle \wedge b$, if $n = m = 0$
   (b) $(O_1 \setminus o_n, O_2) \cdot o_n(b)$, if o_n is an assignment
   (c) $(O_1, O_2 \setminus o'_m) \cdot o'_m(b)$, if o'_m is an assignment
   (d) $((O_1 \setminus o_n), (O_2 \setminus o'_m)) \cdot b(z/x, z/y)$, if $o_n = \alpha?x,\ o'_m = \alpha?y$, and $b(z/x, z/y)$ is a formula obtained from b replacing all occurrences of x and y on a fresh variable z (i.e. z is not occurred in O_1, O_2 and b)
   (e) $((O_1 \setminus o_n), (O_2 \setminus o'_m)) \cdot ((e_1 = e_2) \wedge b)$, if $o_n = \alpha!e_1$ and $o'_m = \alpha!e_2$
   (f) $\perp$, otherwise.

Theorem 1

Let $P_i = (S_{P_i}, s^0_{P_i}, T_{P_i}, \langle P_i \rangle)$ $(i = 1, 2)$ be processes such that $S_{P_1} \cap S_{P_2} = \emptyset$ and $X_{P_1} \cap X_{P_2} = \emptyset$. Then $P_1 \approx P_2$, if there exist a set $\{b_{s_1 s_2} \mid s_i \in S_{P_i}\ (i = 1, 2)\}$ of formulas with variables from $(X_{P_1} \cup X_{P_2}) \setminus \{at_{P_1}, at_{P_2}\}$, such that

1. $\langle P_1 \rangle \wedge \langle P_2 \rangle \leq b_{s^0_{P_1} s^0_{P_2}}$
2. $\forall\, (s_1 \xrightarrow{O} s'_1) \in T_{P_1},\ \forall\, s_2 \in S_{P_2}$ there exists a set $\{s_2 \xrightarrow{T_i} s^i_2 \mid i \in \Im\}$ of CTs of P_2 such that $b_{s_1 s_2} \wedge\ \langle O \rangle \leq \bigvee\limits_{i \in \Im} (O, O_{T_i}) \cdot b_{s'_1 s^i_2}$
3. $\forall\, (s_2 \xrightarrow{O} s'_2) \in T_{P_2},\ \forall\, s_1 \in S_{P_1}$ there exists a set $\{s_1 \xrightarrow{T_i} s^i_1 \mid i \in \Im\}$ of CTs of P_1 such that $b_{s_1 s_2} \wedge\ \langle O \rangle \leq \bigvee\limits_{i \in \Im} (O_{T_i}, O) \cdot b_{s^i_1 s'_2}$

Proof.

Since $X_{P_1} \cap X_{P_2} = \emptyset$, then there is a natural bijection between $X^\bullet_{P_1} \times X^\bullet_{P_2}$ and $(X_{P_1} \cup X_{P_2})^\bullet$. Below we identify these two sets.

We define the relation $\mu \subseteq S^r_{P_1} \times S^r_{P_2}$ as follows:

$$\mu \overset{\text{def}}{=} \{(\xi_1, \xi_2) \in X^\bullet_{P_1} \times X^\bullet_{P_2} \mid b^{(\xi_1, \xi_2)}_{at^{\xi_1}_{P_1} at^{\xi_2}_{P_2}} = 1\} \cup \{(P^0_1, P^0_2)\}.$$

We prove that μ satisfies the conditions from section 8.1.

1. The condition $(P^0_1, P^0_2) \in \mu$ follows from the definition of μ.
2. Let $(v_1, v_2) \in \mu$ and $v_1 \xrightarrow{\tau} v'_1$. We must prove that
$$\exists\, v'_2 : v_2 \xrightarrow{\tau^*} v'_2,\ (v'_1, v'_2) \in \mu \tag{9}$$

We consider separately the cases $v_1 = P_1^0$ and $v_1 \neq P_1^0$.

If $v_1 = P_1^0$, then $v_2 = P_2^0$, and according to definition of the graph P_1^r (section 7.3), $\exists\, \xi_1 \in X^\bullet_{P_1} : \langle P_1 \rangle^{\xi_1} = 1$ and the graph P_1^r has the edge $\xi_1 \overset{\tau}{\to} \xi'_1 = v'_1$, i.e. (ξ_1, τ, ξ'_1) is a realization of a transition $s^0_{P_1} \overset{O_1}{\to} s'_1$ from T_{P_1}, where O_1 is an internal operator.

According to item 2 in the theorem, there exists a set $\{s^0_{P_2} \overset{T_i}{\to} s^i_2 \mid i \in \Im\}$ of CTs of process P_2, such that

$$b_{s^0_{P_1} s^0_{P_2}} \wedge \ \langle O_1 \rangle \leq \bigvee_{i \in \Im} (O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} \tag{10}$$

Since $\langle P_2 \rangle \neq \bot$, then $\exists\, \xi_2 \in X^\bullet_{P_2}$: $\langle P_2 \rangle^{\xi_2} = 1$, so

$$1 = \langle P_1 \rangle^{\xi_1} \wedge \langle P_2 \rangle^{\xi_2} = (\langle P_1 \rangle \wedge \langle P_2 \rangle)^{(\xi_1, \xi_2)} \leq b^{(\xi_1, \xi_2)}_{s^0_{P_1} s^0_{P_2}} \tag{11}$$

(the last inequality holds according to property 1 in the statement of the theorem).

According to the definition of a realization of a transition, the equality $\langle O_1 \rangle^{\xi_1} = 1$ holds. This equality, (10) and (11), imply that there is $i \in \Im$ such that

$$\Big((O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} \Big)^{(\xi_1, \xi_2)} = 1 \tag{12}$$

It is easy to prove that the equality

$$\Big((O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} \Big)^{(\xi_1, \xi_2)} = b_{s'_1 s^i_2}{}^{(\xi_1 \cdot O_1, \xi_2 \cdot O_{T_i})} \tag{13}$$

holds. This equality is an analogue of the equality in the end of section 3.3, and is proved by induction on the total number of AOs in $[O_1]$ and $[O_2]$.

(12) and (13) imply that

$$b_{s'_1 s^i_2}{}^{(\xi_1 \cdot O_1, \xi_2 \cdot O_{T_i})} = 1 \tag{14}$$

By the definition of μ and ξ_2, the statement (9) in this case ($v_1 = P_1^0$) follows from the statement

$$\exists\, \xi'_2 : \xi_2 \overset{\tau^*}{\to} \xi'_2,\ b^{(\xi'_1, \xi'_2)}_{at^{\xi'_1}_{P_1} at^{\xi'_2}_{P_2}} = 1 \tag{15}$$

Define $\xi'_2 \overset{\text{def}}{=} (\xi_2 \cdot (at_{P_2} := s^i_2)) \cdot O_{T_i}$. Since $at^{\xi'_1}_{P_1} = s'_1$, and $\xi'_1 = (\xi_1 \cdot (at_{P_1} := s'_1)) \cdot O_1$, then (15) follows from the statements

$$\xi_2 \overset{\tau^*}{\to} (\xi_2 \cdot (at_{P_2} := s^i_2)) \cdot O_{T_i} \tag{16}$$

$$b^{((\xi_1 \cdot (at_{P_1} := s'_1)) \cdot O_1, (\xi_2 \cdot (at_{P_2} := s^i_2)) \cdot O_{T_i})}_{s'_1 s^i_2} = 1 \tag{17}$$

(16) follows from the definitions of concepts of a CT and a concatenation of operators and from the statements $at^{\xi_2}_{P_2} = s^0_{P_2}$ and $\langle O_{T_i} \rangle^{\xi_2} = 1$. The first

of these statements follows from the equality $\langle P_2\rangle^{\xi_2} = 1$, and the second is justified as follows. The definition of formulas of the form $(O_1, O_2)\cdot b$ implies that the statement (12) can be rewritten as

$$\Big(\langle O_1\rangle \wedge \langle O_{T_i}\rangle \wedge b\Big)^{(\xi_1,\xi_2)} = 1 \tag{18}$$

where b is some formula. Since $X_{P_1} \cap X_{P_2} = \emptyset$, then (18) implies the desired statement $\langle O_{T_i}\rangle^{\xi_2} = 1$.

(17) follows from (14) and from the assumption that at_{P_1} and at_{P_2} do not occur in $b_{s'_1 s^i_2}$, O_1 and O_{T_i}.

Thus, in the case $v_1 = P_1^0$ the property (9) holds.

In the case $v_1 \neq P_1^0$ the property (9) can be proved similarly.

3. Let $(v_1, v_2) \in \mu$ and $v_1 \overset{a}{\to} v'_1$, where $a \neq \tau$. We must prove that

$$\exists\, v'_2 : v_2 \overset{\tau^* a\tau^*}{\longrightarrow} v'_2,\ (v'_1, v'_2) \in \mu \tag{19}$$

(a) At first consider the case $v_1 = P_1^0$ and $a = \alpha?d$.

If $v_1 = P_1^0$, then $v_2 = P_2^0$, and according to the definition of the graph P_1^r (section 7.3), $\exists\, \xi_1 \in X^\bullet_{P_1} : \langle P_1\rangle^{\xi_1} = 1$ and the graph P_1^r has the edge $\xi_1 \overset{a}{\to} \xi'_1 = v'_1$, i.e. (ξ_1, a, ξ'_1) is a realization of a transition t of the form $s^0_{P_1} \overset{O_1}{\to} s'_1$ from T_{P_1}, where O_1 is an input operator. Using the notation introduced at the end of section 7.2, we can write $\xi'_1 = \xi_1 \cdot t^d$.

Just as in the preceding item, we prove that $\exists\, \xi_2 \in X^\bullet_{P_2}$: $\langle P_2\rangle^{\xi_2} = 1$, and there exists a CT $s^0_{P_2} \overset{T_i}{\to} s^i_2$ of the process P_2, such that the equality

$$\Big((O_1, O_{T_i}) \cdot b_{s'_1 s^i_2}\Big)^{(\xi_1,\xi_2)} = 1 \tag{20}$$

holds, which should be understood in the following sense: for each of valuation $\xi \in (X_{P_1} \cup X_{P_2} \cup \{z\})^\bullet$ (where z is a variable, referred in the item 3d of the definition from section 8.2, we can assume that $z \not\in ((X_{P_1} \cup X_{P_2}))$, coinciding with ξ_i on X_{P_i} $(i = 1, 2)$, the equality $\Big((O_1, O_{T_i})\cdot b_{s'_1 s^i_2}\Big)^{\xi} = 1$ holds. In particular, (20) implies that O_{T_i} is an input operator, and $N_{O_{T_i}} = N_{O_1} = \alpha$.

Define $\xi'_2 \overset{\text{def}}{=} \xi_2 \cdot T_i^d$. It is easy to prove that $\xi_2 \overset{\tau^* a\tau^*}{\to} \xi'_2$, and the statement (19) in the case $v_1 = P_1^0$ follows from the equality

$$b^{(\xi_1\cdot t^d, \xi_2\cdot T_i^d)}_{s'_1 s^i_2} = 1 \tag{21}$$

which is justified as follows.

In this case O_1 and O_{T_i} can be represented as concatenation of the form

$$O_1 = (O'_1 \cdot [\alpha?x]) \cdot O''_1, \quad O_{T_i} = (O'_{T_i} \cdot [\alpha?y]) \cdot O''_{T_i}$$

Definition of formulas of the form (8) implies that

$$\begin{array}{l} (O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} = \\ = \Big((O'_1 \cdot [\alpha?x]) \cdot O''_1, (O'_{T_i} \cdot [\alpha?y]) \cdot O''_{T_i} \Big) \cdot b_{s'_1 s^i_2} = \\ = \Big(O'_1 \cdot [\alpha?x], O'_{T_i} \cdot [\alpha?y] \Big) \cdot \Big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \Big) = \\ = (O'_1, O'_{T_i}) \cdot \Big(\big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \big)(z/x, z/y) \Big) \end{array} \tag{22}$$

(20) and (22) imply the equality

$$\Big(\big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \big)(z/x, z/y) \Big)^{(\xi_1 \cdot O'_1, \xi_2 \cdot O'_{T_i})} = 1$$

Its special case is the equality

$$\Big(\big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \big)(d/x, d/y) \Big)^{(\xi_1 \cdot O'_1, \xi_2 \cdot O'_{T_i})} = 1$$

The last equality can be rewritten as

$$\Big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \Big)^{(\xi_1 \cdot O'_1 \cdot (x:=d), \xi_2 \cdot O'_{T_i} \cdot (y:=d))} = 1$$

whence it follows that

$$\Big(b_{s'_1 s^i_2} \Big)^{(\xi_1 \cdot O'_1 \cdot (x:=d) \cdot O''_1, \xi_2 \cdot O'_{T_i} \cdot (y:=d) \cdot O''_{T_i})} = 1 \tag{23}$$

It is easy to see that the left side of (23) coincides with the left side of the equality (21).
Thus, in the case $v_1 = P_1^0$ and $a = \alpha?d$ the property (19) is proven.
In the case $v_1 \neq P_1^0$ and $a = \alpha?d$ the property (19) can be proved similarly.

(b) Now we prove (19), when $a = \alpha!d$. As in the previous item, we consider only the case $v_1 = P_1^0$.
If $v_1 = P_1^0$, then $v_2 = P_2^0$, and

- $\exists\, \xi_1 \in X^\bullet_{P_1} : \langle P_1 \rangle^{\xi_1} = 1$ and the graph P_1^r has the edge $\xi_1 \overset{a}{\to} \xi'_1 = v'_1$, i.e. (ξ_1, a, ξ'_1) is a realization of a transition $t \in T_{P_1}$ of the form $s^0_{P_1} \overset{O_1}{\to} s'_1$, where O_1 is an output operator
- $\exists\, \xi_2 \in X^\bullet_{P_2}$: $\langle P_2 \rangle^{\xi_2} = 1$, and there exists a CT $s^0_{P_2} \overset{T_i}{\to} s^i_2$ of the process P_2, such that

$$\Big((O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} \Big)^{(\xi_1, \xi_2)} = 1 \tag{24}$$

(24) implies that O_{T_i} is an output operator, and $N_{O_{T_i}} = N_{O_1} = \alpha$.
Define $\xi'_2 \overset{\text{def}}{=} \xi_2 \cdot T_i$. For a proof of (19) it is enough to prove the statements

$$\xi_2 \overset{\tau^* a \tau^*}{\to} \xi'_2 \tag{25}$$

$$b^{(\xi_1\cdot t,\xi_2\cdot T_i)}_{s'_1 s^i_2} = 1 \tag{26}$$

In this case O_1 and O_{T_i} can be represented as concatenations of the form

$$O_1 = (O'_1 \cdot [\alpha!e_1]) \cdot O''_1 \tag{27}$$

$$O_{T_i} = (O'_{T_i} \cdot [\alpha!e_2]) \cdot O''_{T_i} \tag{28}$$

The definition of formulas of the form (8) implies that

$$\begin{aligned}
&(O_1, O_{T_i}) \cdot b_{s'_1 s^i_2} = \\
&= \Big((O'_1 \cdot [\alpha!e_1]) \cdot O''_1, (O'_{T_i} \cdot [\alpha!e_2]) \cdot O''_{T_i}\Big) \cdot b_{s'_1 s^i_2} = \\
&= \Big(O'_1 \cdot [\alpha!e_1], O'_{T_i} \cdot [\alpha!e_2]\Big) \cdot \Big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2}\Big) = \\
&= (O'_1, O'_{T_i}) \cdot \left\{\begin{array}{l} e_1 = e_2 \\ (O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \end{array}\right\}
\end{aligned} \tag{29}$$

(24) and (29) imply the equality

$$\left\{\begin{array}{l} e_1 = e_2 \\ (O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2} \end{array}\right\}^{(\xi_1\cdot O'_1, \xi_2\cdot O'_{T_i})} = 1$$

from which it follows that

$$e_1^{\xi_1\cdot O'_1} = e_2^{\xi_2\cdot O'_{T_i}} \tag{30}$$

$$\Big((O''_1, O''_{T_i}) \cdot b_{s'_1 s^i_2}\Big)^{(\xi_1\cdot O'_1, \xi_2\cdot O'_{T_i})} = 1 \tag{31}$$

By assumption, $(\xi_1, \alpha!d, \xi'_1)$ is a realization of the transition $s^0_{P_1} \xrightarrow{O_1} s'_1$. From the representation of O_1 as a concatenation (27) it follows that $d = e_1^{\xi_1\cdot O'_1}$, whence, according to (30) we get the equality $d = e_2^{\xi_2\cdot O'_{T_i}}$. From this and from a representation of O_{T_i} as a concatenation (28) it follows that $(\xi_2, \alpha!d, \xi_2\cdot T_i)$ is a realization of the CT T_i. Since $\xi_2\cdot T_i = \xi'_2$ and $\alpha!d = a$, then it follows that we are justified the statement (25).

The statement (26) follows from (31).

Thus, in the case $v_1 = P^0_1$ and $a = \alpha!d$ the property (19) is proven.

In the case $v_1 \neq P^0_1$ and $a = \alpha!d$ the property (19) can be proved similarly

The symmetrical conditions on the relation μ (i.e., second parts of the conditions on μ, presented in second and third items in section 8.1) can be proved similarly. ■

9 Simplification of processes

9.1 A concept of a simplification of processes

The concept of a simplification of processes is intended to reduce the problem of verification of processes.

A **simplification** of a process P is a sequence of transformations of this process, each of which is performed according to one of the rules set out below. Each of these transformations (except the first) is performed on the result of previous transformation. A **result** of a simplification is a result of last of these transformations.

Simplification rules are defined as follows. Let P be a process.

Rule 1 (removing of states).
If $s \in S_P \setminus \{s^0_P\}$, and

- $s_1 \overset{O_1}{\to} s$, …, $s_n \overset{O_n}{\to} s$ are all transitions incoming to s
- $s \overset{O'_1}{\to} s'_1$, …, $s \overset{O'_m}{\to} s'_m$ are all transitions outgoing from s, and if all these transitions are internal, then $\langle O'_i \rangle \wedge \langle O'_j \rangle = \perp$ if $i \neq j$
- $s \not\in \{s_1, \ldots, s_n, s'_1, \ldots, s'_m\}$
- $\forall\, i = 1, \ldots, n,\ \forall\, j = 1, \ldots, m \quad \exists\, O_i \cdot O'_j$

then s and all transitions related to s are removed from P, and the transitions $s_i \overset{O_i \cdot O'_j}{\longrightarrow} s'_j$ (where $i = 1, \ldots, n,\ j = 1, \ldots, m$) are added to P.

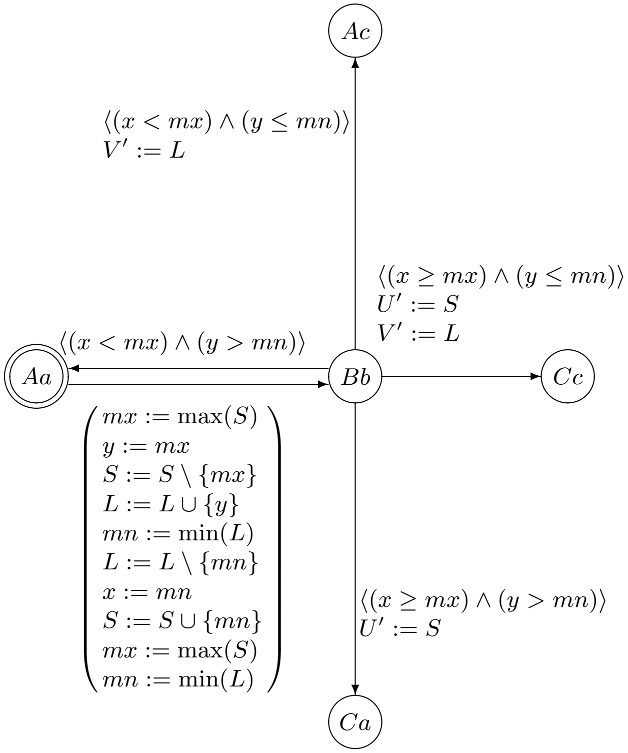
Rule 2 (fusion).
If P has a pair of transitions of the form $s_1 \overset{O}{\to} s_2$, $s_1 \overset{O'}{\to} s_2$, and $[O] = [O']$, then this pair is replaced by a single transition $s_1 \overset{b[O]}{\longrightarrow} s_2$, where $b = \langle O \rangle \vee \langle O' \rangle$.

Rule 3 (elimination of unessential assignments).
If P has an AO $(x := e)$, where $x \not\in X^s_P$, then this AO is removed from P.

Theorem 2
If P' is a result of simplification of P, then $P' \approx P$.

9.2 An example of a process obtained by a simplification

In this section we present a process which is obtained by a simplification of the process (2). This process has the following form:


Ac
$\langle (x < mx) \wedge (y \leq mn) \rangle$
$V' := L$
$\langle (x \geq mx) \wedge (y \leq mn) \rangle$
$U' := S$
$V' := L$
$\langle (x < mx) \wedge (y > mn) \rangle$
Aa
Bb
Cc
$mx := \max(S)$
$y := mx$
$S := S \setminus \{mx\}$
$L := L \cup \{y\}$
$mn := \min(L)$
$L := L \setminus \{mn\}$
$x := mn$
$S := S \cup \{mn\}$
$mx := \max(S)$
$mn := \min(L)$
$\langle (x \geq mx) \wedge (y > mn) \rangle$
$U' := S$
Ca

(32)

This simplified process allows to detect some simple flaws of the algorithm of separation of sets, for examle a possibility of a deadlock situation: there are states of the process (32) (namely, Ac and Ca) such that

- there is no transitions starting at these states
- but falling into these states is not a normal completion of the process.

9.3 Another example of a simplification of a process

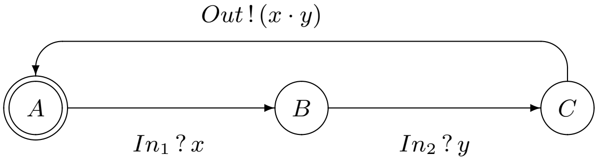
Suppose we have a system “multiplier”, which has

- two input ports with the names In_1 and In_2, and
- one output port with the name Out.

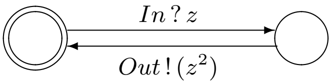
An execution of the multiplier is that it

- receives on its input ports two values, and
- gives their product on the output port.

A behavior of the multiplier is described by the process Mul:



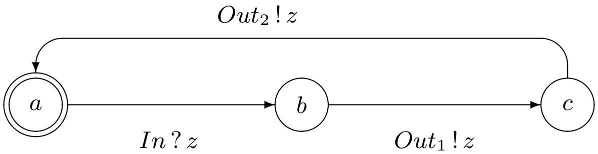
Using this multiplier, we want to build a system "calculator of a square", whose behavior is described by the process $Square\_Spec$:



The desired system is a composition of

- the auxiliary system "duplicator" having
  - an input port In, and
  - output ports Out_1 and Out_2

  behavior of which is described by the process Dup:

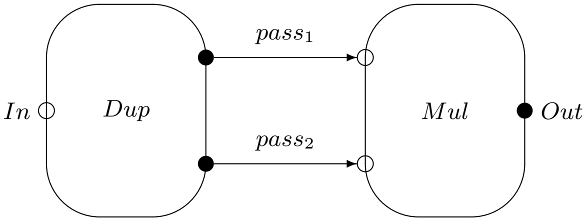


  i.e. the duplicator copies its input to two outputs, and
- the multiplier, which receives on its input ports those values that duplicator gives.

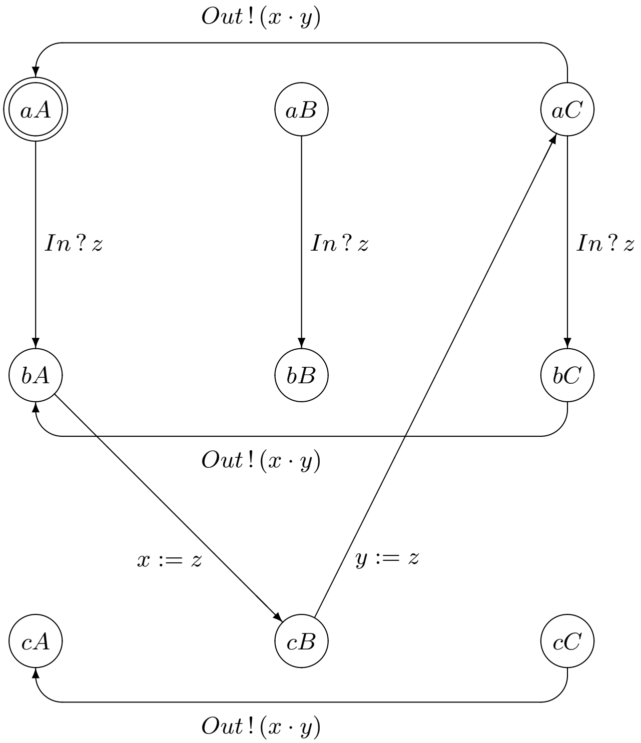
A process $Square$, corresponding to such a composition is defined as follows:

$$Square \overset{\text{def}}{=}$$
$$\overset{\text{def}}{=} \left(\begin{array}{l} Dup[pass_1/Out_1, pass_2/Out_2] \ \mid \\ \mid \ Mul[pass_1/In_1, pass_2/In_2] \end{array} \right) \setminus \{pass_1, pass_2\}$$

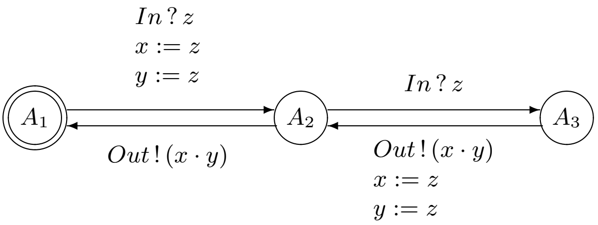
A flow graph of the process $Square$ has the form



However, the process *Square* does not meet the specification *Square_Spec* (i.e. *Square* and *Square_Spec* are not observationally equivalent). This fact is easy to detect by a construction of a graph representation of *Square*, which, by definition of operations of parallel composition, restriction and renaming, is the following:



After a simplification of this process we obtain the process



(33)

which shows that

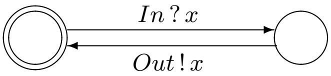
- the process *Square* can execute two input actions together (i.e. without an execution of an output action between them), and

– the process *Square_Spec* can not do that.

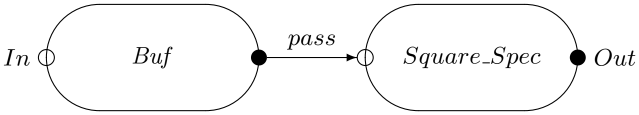
The process *Square* meets another specification:

$$Square\_Spec' \stackrel{\text{def}}{=} \left(\begin{array}{l} Buf[pass/Out] \ \mid \\ \mid Square\_Spec[pass/In] \end{array} \right) \setminus \{pass\}$$

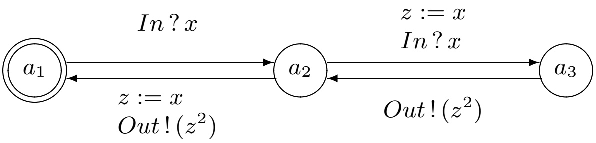
where *Buf* is a buffer which can store one message, whose behavior is represented by the diagram



A flow graph of *Square_Spec'* has the form



A simplified process *Square_Spec'* has the form

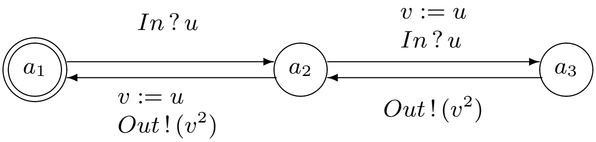


(34)

The statement that *Square* meets the specification *Square_Spec'* can be formalized as

$$(33) \approx (34) \tag{35}$$

We justify (35) with use of theorem 1. At first, we rename variables of the process (34), i.e. instead of (34) we shall consider the process



(36)

To prove (33) $\approx$ (36) with use of theorem 1 we define the formulas b_{A_i,a_j} (where $i, j = 1, 2, 3$) as follows:

- $b_{A_i,a_j} \stackrel{\text{def}}{=} \perp$, if $i \neq j$
- $b_{A_1,a_1} \stackrel{\text{def}}{=} \top$
- $b_{A_2,a_2} \stackrel{\text{def}}{=} (x = y = z = u)$
- $b_{A_3,a_3} \stackrel{\text{def}}{=} (x = y = v) \wedge (z = u)$.

10 An example: verification of a sliding window protocol

In this section we present an example of use of theorem 1 for a verification of a sliding window protocol.

A sliding window protocol ensures a transmission of messages from one agent to another through a medium, in which messages may get distorted or lost. In this section we consider a two-way sliding window protocol, in which the agents can both send and receive messages from each other. We do not present here a detail explanation of this protocol, a reader can find it in section 3.4.2 of the book [10] (a protocol using go back n).

10.1 A structure of the protocol

The protocol is a system consisting of interacting components, including

- components that perform a formation, sending, receiving and processing of messages (such components are called **agents**, and messages sent from one agent to another, are called **frames**), and
- a medium, through which frames are forwarded (such a medium is called a **channel**).

A detailed description of the components and relation between them is represented in the Appendix.

10.2 Frames

Each frame f, which is sent by any of the agents, contains a packet x, and a couple of numbers:

- a number $s \in \mathbf{Z}_n \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1\}$ (where n is a fixed integer), which is associated with the packet x and with the frame f, and
- a number $r \in \mathbf{Z}_n$, which is a number associated with a last received undistorted frame.

To build a frame, a function φ is used, i.e. a frame has the form $\varphi(x, s, r)$.

To extract the components x, s, r from the frame $\varphi(x, s, r)$, the functions $info$, seq and ack are used, these functions have the following properties:

$$info(\varphi(x, s, r)) = x, \quad seq(\varphi(x, s, r)) = s, \quad ack(\varphi(x, s, r)) = r$$

10.3 Window

The set of variables of an agent contains an array $x[n]$. Values of some components of this array are packets which are sent, but not yet acknowledged. A set of components of the array x, which contain such packets at a current time, is called a **window**.

Three variables of the agent are related to the window: b (a lower bound of the window), s (an upper bound of the window), and w (a number of packets in the window). Values of these variables belong to the set $\mathbf{Z}_n$. At the initial moment values of b, s and w are equal to 0. At any moment the window can be empty (if $b = s$), or not empty (if $b \neq s$). In the last case the window consists of elements of x with indices from the set $[b, s[$, where $[b, s[$ denotes the set

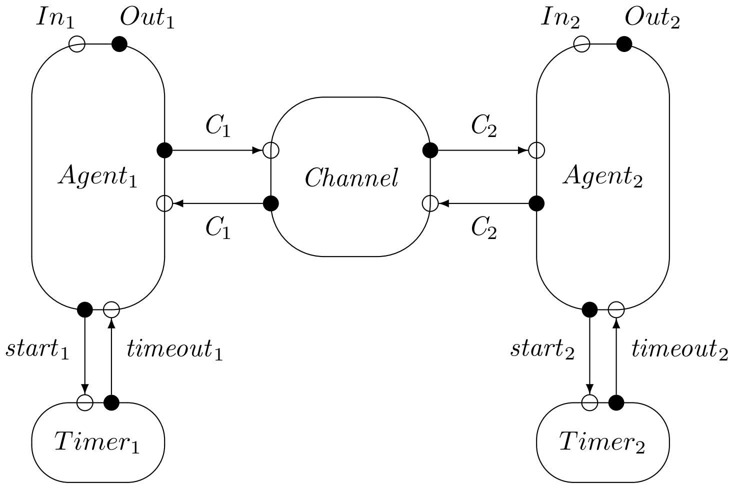
- $\{b, b+1, \ldots, s-1\}$, if $b < s$, and
- $\{b, b+1, \ldots, n\} \cup \{0, 1, \ldots, s-1\}$, if $s < b$.

Adding a new packet to the window is performed by an execution of the following actions: this packet is written in the component $x[s]$, s is increased by 1 modulo n (i.e. a new value of s is assumed to be $s+1$, if $s < n-1$, and 0, if $s = n-1$), and w is increased by 1. Removing a packet from the window is performed by an execution of the following operations: b is increased by 1 modulo n, and w is decreased by 1 (i.e. it is removed a packet whose number is equal to the lower bound of the window).

If an agent received a frame, the third component r of which (i.e. a number of an acknowledgment) is such that $r \in [b, s[$, then all packets in the window with numbers from $[b, r[$ are considered as acknowledged and are removed from the window (even if their acknowledgments were not received).

10.4 Flow graph

A relation between subprocesses of sliding window protocol is represented by the flow graph:



10.5 Timers

Each component $x[i]$ of the array x is associated with a timer, which determines a duration of waiting of an acknowledgement from another agent of a receiving of the packet contained in the component $x[i]$. The combination of these timers is considered as a process $Timer$, which has an array $t[n]$ of boolean variables. The process $Timer$ has one state and transitions which are labeled by the following operators:

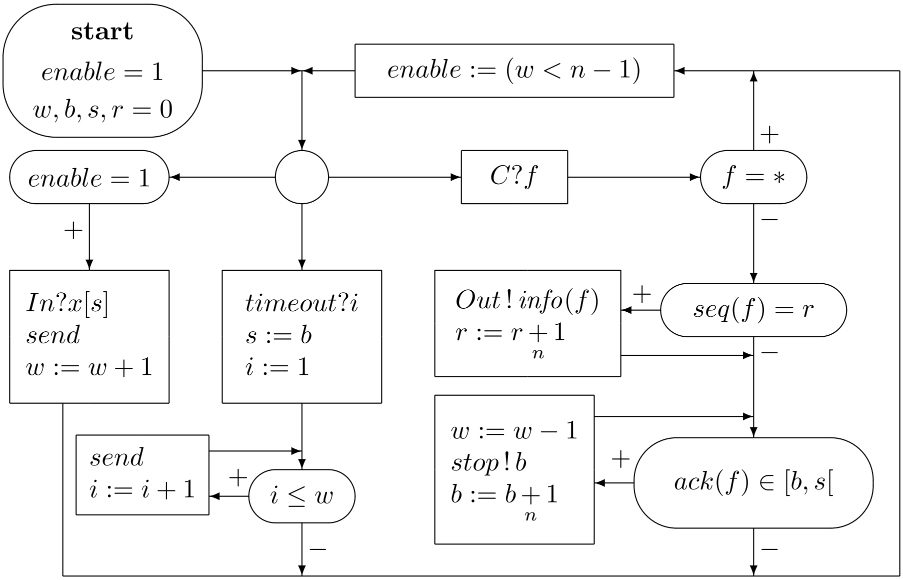
- $[start?i, t\,[i] := 1]$
- $[stop?i,\ t\,[i] := 0]$
- $(t\,[j] = 1)[timeout\,!\,j, t\,[j] := 0]$ (where $j = 0, \ldots, n-1$)

An initial condition is $t = (0, \ldots, 0)$.

If an agent has received an object with a name *timeout* from a timer, then the agent sends again all packets from its window.

10.6 Agents

A behavior of each agent is described by the same process, combining functions of a sender and a receiver. This behavior can be represented by the following flowchart.



where

- *send* is an abbreviation of the list of AOs $\left\{\begin{array}{l} C\,!\,\varphi(x[s], s, r \underset{n}{-} 1) \\ start\,!\,s \\ s := s \underset{n}{+} 1 \end{array}\right\}$

- $*$ is a special notation for a distorted message, and
- a value of the variable *enable* is 1, if the agent can receive a new packet from his network level (i.e. $w < n-1$), and 0, otherwise.

Processes $Agent_1$ and $Agent_2$ are obtained by a simple transformation of this flowchart, and by an addition of corresponded index (1 or 2) to its variables and names.

10.7 Specification

External actions of the above protocol (i.e. actions which are related to its communication with a network level) have the form $In_1?d$, $In_2?d$, $Out_1!d$ and $Out_2!d$. Assume that we take into account only external actions $In_1?d$ and $Out_2!d$, and ignore other its external actions (i.e. we consider a transmission only in one direction: from the left to the right). We would like to prove that such behavior is equivalent to a behavior of a process B_{n-1}, which is called "a FIFO buffer which can hold at most $n-1$ frames", and is defined as follows:

- variables of B_{n-1} are
  - an array $(x[0], \ldots, x[n-1])$, elements of which have the same type as a type of frames in the above protocol, and
  - variables r, s, u, values of which belong to $\mathbf{Z}_n$, and have the following meaning: at every moment
    - $*$ a value of u is equal to a number of frames in the buffer
    - $*$ values r and s can be interpreted as lower and upper bounds of a part of the array x, which stores the received frames, which has not yet been issued from the buffer
- B_{n-1} has one state and 2 transitions with labels

$$\begin{array}{l} (u < n-1)\,[In?x[s],\ s := s \underset{n}{+} 1,\ u := u+1] \\ (u > 0)\ [Out\,!\,x[r],\ r := r \underset{n}{+} 1,\ u := u-1] \end{array}$$

  where $\forall\, i \in \{0, n-2\}\ \ i \underset{n}{+} 1 \overset{\text{def}}{=} i{+}1$ and $(n-1) \underset{n}{+} 1 \overset{\text{def}}{=} 0$
- initial condition is $r = s = u = 0$.

10.8 A process corresponded to the protocol

A process that describes a behavior of the protocol with respect to the above specific point of view (where we ignore actions of the form $In_2?d$ and $Out_1!d$) is constructed as a parallel composition of the processes corresponded to components of this procotol, with elimination of atomic operators related to ignored communications.

10.9 Verification

With use of the simplification operations from section 9, we can transform the process corresponded to the protocol (with elimination of atomic operators which are corresponded to ignored actions) to a process P with only one state and with transitions labelled by the following operators:

- $(w < n-1)\ [In?x[s],\ M_1 := M_1 \cdot \varphi(x[s], s, \ldots),\ s := s \underset{n}{+} 1,\ w := w+1]$
- $(M_1 \neq \varepsilon) \wedge (seq(\hat{M}_1) = r)\ [Out!info(\hat{M}_1),\ r := r \underset{n}{+} 1,\ M_1 := M_1']$
- $(M_2 \neq \varepsilon) \wedge (ack(\hat{M}_2) \in [b, s[)\ [b := ack(\hat{M}_2) \underset{n}{+} 1,\ w := s \underset{n}{-} b,\ M_2 := M_2']$, where $\forall\, i, j \in \{0, n-1\}$ $\ i \underset{n}{-} j \overset{\text{def}}{=} i-j$, if $i-j \in \{0, n-1\}$, and $n+i-j$, otherwise
- $[M_1 := M_1 \cdot \varphi(x[b], b, \ldots), \ldots , M_1 := M_1 \cdot \varphi(x[s \underset{n}{-} 1], s \underset{n}{-} 1, \ldots)]$
- $(M_1 \neq \varepsilon)\ [M_1 := M_1']$
- $(M_2 \neq \varepsilon)\ [M_2 := M_2']$
- $[M_2 := M_2 \cdot \varphi(\ldots, \ldots, r \underset{n}{-} 1)]$

where

dots denote unessential components of expressions, and the symbols M_i, $\hat{M}_i$, M_i', $\cdot$ and ε have the following sense:

- M_1 and M_2 are variables of the process *Channel*, and values of these variables are lists of frames which were received by the process *Channel* (M_i holds frames received from $Agent_i$), every received frame is added to the end of a corresponded list
- $\hat{M}_i$ $(i = 1, 2)$ is an expression, a value of which is equal to the first element of the list M_i
- M_i' $(i = 1, 2)$ is an expression, a value of which is equal to the list M_i without its first element
- $\cdot$ is a function of an addition of a frame to the end of a list
- ε is a constant, a value of which is an empty list.

For a proof that the process P is observationally equivalent to the process B_{n-1}, we define a formula $b_{s_1 s_2}$ where s_1 is a unique state of P and s_2 is a unique state of B_{n-1} as a conjunction of the following formulas:

- $(M_1 \neq \varepsilon) \wedge (seq(\hat{M}) = r) \ \Rightarrow\ u > 0$
- $\forall\, f \in M_1\ info(f) = x[seq(f)]$
- $\forall\, f \in M_2 \quad ack(f) \in [b \underset{n}{-} 1, r[$
- $[r, s[\ \subseteq\ [b, s[$
- $w = s \underset{n}{-} b \leq n-1$
- $u = s \underset{n}{-} r \leq w$
- if a value of M_2 is $f_1 \cdot \ldots \cdot f_k$, then the sequence $ack(f_1) \quad \ldots \quad ack(f_k)$ is monotonically increasing (mod n) subsequence of $[b \underset{n}{-} 1, r[$

(the last record is not a formula, but can be represented by a formula, we omit this representation).

It is not so diffcult to check that $b_{s_1 s_2}$ satisfies the conditions of theorem 1 and this proves that the process P is observationally equivalent to B_{n-1}.

11 Conclusion

The concept of a process with message passing which is presented in this paper can be considered as a formal model of a communicating program without recursion. In the paper we have established suffcient conditions of observational equivalence of processes. The next steps of investigations in this area can be the following.

- Find necessary and suffcient conditions of observational equivalence of processes with message passing.
- Generalize the proposed concept of a process with message passing for formal modeling of communicating programs with recursion, and find necessary and suffcient conditions of observational equivalence of such processes.

References

1. R. Milner: A Calculus of Communicating Systems. Number 92 in Lecture Notes in Computer Science. Springer Verlag (1980)
2. R. Milner: Communicating and Mobile Systems: the π-calculus. Cambridge University Press (1999)
3. C.A.R. Hoare: Communicating Sequential Processes. Prentice Hall (1985)
4. Clarke, E.M., Grumberg, O., and Peled, D.: Model Checking, MIT Press (1999)
5. C.A. Petri: Introduction to general net theory. In W. Brauer, editor, Proc. Advanced Course on General Net Theory, Processes and Systems, number 84 in LNCS, Springer Verlag (1980)
6. J.A. Bergstra, A. Ponse, and S.A. Smolka, editors: Handbook of Process Algebra. North-Holland, Amsterdam (2001)
7. D. Brand, P. Zafiropulo: On Communicating Finite-State Machines. Journal of the ACM, Volume 30 Issue 2, April 1983, pp. 323-342. ACM New York, NY, USA (1983)
8. R.W. Floyd: Assigning meanings to programs. In J.T. Schwartz, editor, Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science, pages 19-32. AMS (1967)
9. Badban, B. and Fokkink, W.J. and van de Pol, J.C.: Mechanical Verification of a Two-Way Sliding Window Protocol (Full version including proofs). Internal Report TR-CTIT-08-45, Centre for Telematics and Information Technology, University of Twente, Enschede, June 2008. `http://doc.utwente.nl/64845/` (2008)
10. A. Tanenbaum: Computer Networks. Fourth Edition. Prentice Hall (2002)
11. B. Hailpern: Verifying Concurrent Processes Using Temporal Logic. LNCS 129. Springer-Verlag (1982)
12. G. Holzmann: Design and Validation of Computer Protocols. Prentice Hall (1991)
13. G. Holzmann: The model checker Spin. IEEE Transactions on Software Engineering, 23:279-295 (1997)

14. R. Kaivola: Using compositional preorders in the verification of sliding window protocol. In Proc. 9th Conference on Computer Aided Verification, LNCS 1254, pages 48-59 (1997)
15. P. Godefroid and D. Long: Symbolic protocol verification with Queue BDDs. Formal Methods and System Design, 14(3):257-271 (1999)
16. K. Stahl, K. Baukus, Y. Lakhnech, and M. Steffen: Divide, abstract, and model-check. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, Proc. 6th SPIN Workshop on Practical Aspects of Model Checking, Lecture Notes in Computer Science 1680, pages 57-76. Springer-Verlag (1999)
17. T. Latvala: Model checking LTL properties of high-level Petri nets with fairness constraints. In J. Colom and M. Koutny, editors, Proc. 21st Conference on Application and Theory of Petri Nets, Lecture Notes in Computer Science 2075, pages 242-262. Springer-Verlag (2001)
18. D. Chkliaev, J. Hooman, and E. de Vink: Verification and improvement of the sliding window protocol. In H. Garavel and J. Hatcliff, editors, Proc. 9th Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 2619, pages 113-127 (2003)
19. A. Schoone: Assertional Verification in Distributed Computing. PhD thesis, Utrecht University (1991)
20. F. Vaandrager: Verification of two communication protocols by means of process algebra. Technical Report Report CS-R8608, CWI (1986)
21. R. Groenveld: Verification of a sliding window protocol by means of process algebra. Technical Report P8701, University of Amsterdam (1987)
22. J. van Wamel: A study of a one bit sliding window protocol in ACP. Technical Report P9212, University of Amsterdam (1992)
23. M. Bezem and J. Groote: A correctness proof of a one bit sliding window protocol in μCRL. The Computer Journal, 37(4):289-307 (1994)