ՀԱՅԱՍՏԱՆԻ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ԱԶԳԱՅԻՆ ԱԿԱԴԵՄԻԱ
НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК АРМЕНИИ
NATIONAL ACADEMY OF SCIENCES OF ARMENIA
ДОКЛАДЫ ՁԵԿՈՒՅՑՆԵՐ REPORTS

Հատոր Том Volume 109 2009 № 4

MATHEMATICS

УДК 519.681

H. A. Grigoryan, A. S. Shoukourian

Equivalence of Processes in Object-Oriented Environments with Commutative and Non-Commutative Objects

(Submitted by corresponding member of NAS RA I.D. Zaslavsky 30/IX 2009)

Keywords: *object-oriented environments, model of an environment, processes, equivalence of processes, multitape multidimensional automata, partially commutative semigroups, partially commutative alphabet, regular expressions*

1. Introduction. The concept of a process [1, 2] is fundamental to many areas of science and engineering. The common idea of a process is a pattern of activity performed by distributed entities (often named objects, agents [3]) to achieve particular objectives. As sub-processes are often set up to perform equivalent operations, solutions to problem of deciding the equivalence of processes are of high practical relevance for all applications that involve design, monitoring, and control of processes. Many forms of equivalences (specifically, bisimulation, trace equivalence, functional equivalence [2-5]) have been considered, but the problem for distributed, temporally extended processes has mainly been investigated for process models that do not contain peculiarities of object/agent-oriented environments [3]. Meantime these peculiarities are inherent and essential for modern manufacturing frameworks [6].

This paper addresses a model of manufacturing processes introduced in [3]. An extension of the general model defined over an object-oriented environment is suggested for consideration of functional equivalence problem for processes. The considered formalism does not cover completely all the peculiarities of the model in [3], it is intended for applications where triggering of events for a given situation occurs periodically, according to some cycle. The formalism includes a definition of commutative operations and objects, semantics of process execution and a definition of the functional equivalence problem for processes.

It is shown that, if there are no commutative operations, the problem is reduced to the equivalence problem of multidimensional multitape automata [7] and, thus, is solvable [8, 9].

In the case of commutative operations the equivalence problem for environments with two commutative objects, with two or more operations per each, is insolvable [10].

If there is only one object among commutative objects with two or more operations then it is shown that the equivalence problem is reduced to the equivalence problem of regular expressions over a partially commutative alphabet [11] and, thus, is solvable.

2. Model Equations and Semantics. A formal model of an environment based on a finite set of communicating objects is introduced below. Objects are communicating with each other via a finite set of messages. As there could be several messages passed to a given object, each object has a possibility to gather a finite number of received input messages in a queue before processing. A newly received message is ignored, if it comes after the message queue is filled completely.

Object consists of a unique identifier, a finite set of states and a finite set of operations. Operations are carried out in response to messages, last some time interval, and, for a current state, result in a new state as well as in a vector of messages to be sent to other objects. A state of the environment is a vector of states of contained objects. State and operation sets of different objects within the environment are considered disjoint.

Basic model equations describing environment, object, operation as well as duration of operations and finite message queues are defined as follows:

$$Environment = (o_1, \dots, o_n), \textit{ where } o_i \in \{Object\},\ i = 1, \dots, n$$

$$Object = (ID, \{State\}, \{Operation\})$$

$$MSG = \{msg_e, msg_1, \dots, msg_k\}$$

$$Operation = \{State\} \times MSG \to \{State\} \times \{Reaction\}$$

$$Reaction \in MSG^n,\ Reaction\ [i] \textit{ is the component } i \textit{ of the vector } Reaction,\ i = 1, \dots, n$$

$$Duration = \{Operation\} \to \{1, 2, \dots\}$$

Then the next group of model equations is added to describe message communication within the environment, including message addition to queues for objects of the environment. Each object queue has its own length p_i.

$$EnvState = (s_1, \dots, s_n),\ s_i \in \{State\} \textit{ for the object } o_i,\ i = 1, \dots, n,\ EnvState[i] = s_i$$

$$MSGQueue^{(p)} = \bigcup_{m=0}^{p} (MSG - msg_e)^m,\ p \in \{0, 1, 2, \dots\},\ MSGQueue^{(p)}[i] \textit{ is the component } i \textit{ of the vector } MSGQueue^{(p)}$$

$$EnvInput \in MSGQueue^{(p_1)} \times \dots \times MSGQueue^{(p_n)}$$

Finally, a notion of scene is introduced, which describes the status of the environment at a given time instance t as well as the change of $EnvInput$ basing on a given $Reaction$.

$Scene = (t, EnvState, EnvInput), t \in \{1, 2, \dots\}$

$EnvInput \overset{\bigcirc}{+} Reaction = (EnvInput[1] \overset{\bigcirc}{+} Reaction[1], \dots, EnvInput[n] \overset{\bigcirc}{+} Reaction[n])$

$MSGQueue^{(p)} = (msg^{(1)}, \dots, msg^{(m)})$, *where* $msg^{(1)}, \dots, msg^{(m)} \in (MSG - msg_e)$, $m \leq p$

$MSGQueue^{(p)} \overset{\bigcirc}{+} msg = (msg^{(1)}, \dots, msg^{(m)}, msg)$, *if* $m < p$, $msg \in (MSG - msg_e)$

$MSGQueue^{(p)} \overset{\bigcirc}{+} msg = MSGQueue^{(p)}$, *if* $m = p$

$MSGQueue^{(p)} \overset{\bigcirc}{+} msg_e = MSGQueue^{(p)}$

Any operation op of a given object o can be naturally extended to operation $op^{(ext)} : \{EnvState\} \times \{EnvInput\} \rightarrow \{EnvState\} \times \{EnvInput\}$ in the following way.

$\forall EnvState \forall EnvInput\ op^{(ext)}(EnvState, EnvInput) = (EnvState', EnvInput')$, *where*

$EnvState'[j] = EnvState[j]$, $j \neq i$

$EnvState'[i] = (op(EnvState[i], (EnvInput[i])[1]))[1]$

$EnvInput' = EnvInput \overset{\bigcirc}{+} (op(EnvState[i], (EnvInput[i])[1]))[2]$,

where $(EnvInput[i])[1]$ *is the first message in the queue* $EnvInput[i]$, $(op(EnvState[i], (EnvInput[i])[1]))[v]$, $v = 1, 2$ *are, correspondingly, first and second components of the resulting vector for the source operation* op.

Only extended operations will be considered further. To simplify the notation, op will be used instead of $op^{(ext)}$ meaning the mapping: $EnvStateSet \times EnvInputSet \rightarrow EnvStateSet \times EnvInputSet$.

A binary meta-operation named *concatenation* and denoted by $*$ is defined over extended object operations of a given environment.

Let op_i be an extended operation of an object o_i and op_j be an extended operation of an object o_j. Then $op_i * op_j$ is an operation meaning the mapping $\{EnvState\} \times \{EnvInput\} \rightarrow \{EnvState\} \rightarrow \{EnvInput\}$ in the following way:

$$op_i * op_j(EnvState, EnvInput) = op_j(op_i(EnvState, EnvInput)).$$

Let $Op = \{op_1, \dots, op_k\}$ be a set of all extended operations for all objects of a given environment including also an empty operation which provides a trivial same-to-same mapping $\{EnvState\} \times \{EnvInput\}$ into $\{EnvState\} \times \{EnvInput\}$.

Then, using the introduced meta-operation of concatenation a semi-group of all words in the alphabet Op can be considered. It will be denoted further by F_{Op}.

Two objects o_i and o_j are named *commutative* if and only if for any extended operation op_{i1} of the object o_i and for any extended operation op_{j1} of the object o_j $\forall EnvState\ \forall EnvInput$

$$op_i * op_j(EnvState, EnvInput) = op_j * op_i(EnvState, EnvInput).$$

A notion of process is introduced to provide a possibility of a programmable control over an environment. It is started by introducing the following primitives:

$TimeConstraint = \{t_0 + n * \Delta t | n \in \{0, 1, 2, \dots\}\}$, *where* t_0 *and* Δt *are non-negative integers*

$BasicPred = \{\pi_1, \dots, \pi_r\}$, *where* π_j *is a predicate symbol of arity* n_j

$Condition = \pi_j(o_{i_1}.State, \dots, o_{i_{n_j}}.State)$, *where* $_j \in BasicPred$

$Assertion = (Condition, TimeConstraint)$

$Situation = (Object, Assertion)$

$SituationBatch = \{Situation\}$

Let $\Theta = \{o_1, \dots, o_n\}$ be an environment.

A *process* over an environment Θ is a tuple $P = \langle \Theta, \Sigma, B, \Gamma, b_s \rangle$, where:
Σ - is an ordered finite set of situations, such that $\forall \sigma \in \Sigma$, $\sigma.Object \in \Theta$
B - is a set of situation-batches, such that $\forall b \in B\, b \subseteq \Sigma$, $b_e = \emptyset \in B$ is the end situation batch
Γ - is a relation $\{(\sigma, b, \omega) | \sigma \in \Sigma\}$, where $b \in B$, $\omega \in \sigma.Object.Operatons$
b_s - is the start situation batch.

An *interpretation* is built for a given process, if the initial state of the environment and following functions are defined:

- initial $EnvState : es_0 = (st_0^{(1)}), \dots, st_0^{(n)})$
- a function $f_{op} : \{EnvState\} \times \{EnvInput\} \rightarrow \{EnvState\} \times \{EnvInput\}$ for each operation symbol $op \in Op$
- a function $\rho_{\pi_j} : o_{i_1}.StateSet \times \dots \times o_{i_{n_j}}.StateSet \rightarrow \{TRUE, FALSE\}$ (n_j is the arity of the corresponding predicate symbol) for each predicate symbol.

The semantics of a process interpretation is described below via the following execution algorithm.

To ensure consistent execution of the process, concurrent execution of two and more operations of the same object is not allowed. So, in case if there are two pending situations, ready to be executed at the same time, it must be chosen which one to execute first. The other situation should stay in a pending state. To implement this, the set of situations (Σ) is defined above as ordered, as well as corresponding checks are performed in the execution algorithm.

Data structures and functions used in the execution algorithm are listed below.

- **InputScene** - the initial scene (input of the algorithm)
- **Pending** - a set of situations that are waiting to be executed
- **Running** - a set of situations Σ', for which $\forall \sigma \in \Sigma'$, $operation(\sigma)$ is currently running

- **CurrentEnvInput** - the current environment input
- **nextNode**$(\sigma) = b$, if $(\sigma, b, \omega) \in \Gamma$
- **operation**$(\sigma) = \omega$, if $(\sigma, b, \omega) \in \Gamma$
- **receivedMsg**$(\sigma) = TRUE$, if a message needed for $operation(\sigma)$ is in **CurrentEnvInput** (in the message queue of the corresponding object).
- **canExecute**(σ) =receivedMsg$(\sigma)\&\sigma.Assertion.Condition$, where σ is a situation.
- **beginExecution**(σ) - removes the message needed for $operation(\sigma)$ from **CurrentEnvInput** (from the message queue of the corresponding object).
- **finalizeExecution**(σ) - changes the state of the corresponding object and adds the $ObjectReaction$ of $operation(\sigma)$ to **CurrentEnvInput (CurrentEnvInput = CurrentEnvInput** $\overset{\bigcirc}{+} ObjectReaction$**).**
- **executed**$(\sigma) = TRUE$, if $duration(operation(\sigma))$ time has passed from the corresponding **beginExecution**(σ).
- **add(dest, sitBatch)** - adds all situations of the situation batch **sitBatch** to the set **dest** (no duplications).
- **remove(source, sit)** - removes situation **sit** from the set **source.**
- **move(sit, source, dest)** - removes situation **sit** from the set **source** and adds to the set **dest** (no duplications).
- **foreach**$(\sigma, cond(\sigma))$ - iterates sequentially over any situation σ for which a given condition over $\sigma - cond(\sigma)$ is true, according to the order defined in Σ.

Execution Algorithm

```
Pending = Running = ∅;
CurrentEnvInput = InputScene.EnvInput;
add(Pending, b_s);
for(t = InputScene.t; ; ++t)
begin
  foreach(σ, σ ∈Running & executed(σ))
  begin
    finalizeExecution (σ);
    remove(Running, σ);
    add(Pending, nextNode(σ));
  end
```

```
foreach(σ, σ ∈Pending & canExecute(σ)
 & t ∈ σ.Assertion.TimeConstraint
 & (∀σ1 ∈Running σ1.object≠ σ.object))
begin
 move(σ, Pending, Running);
 beginExecution(σ);
end

if(Running = ∅ & (∀σ ∈Pending¬canExecute(σ)))
 exit;
end
```

The execution is **successful** if **Pending** $= \emptyset$ at the end, otherwise the execution is **failed**. For a successful execution, the result of the algorithm is the last scene before the end of the algorithm (output scene). The tuple ($Pending$, $Running$, $CurrentTime$, $CurrentEnvInput$) will be called an *execution state* of the algorithm.

Two processes defined over the same environment will be called *functionally equivalent* if and only if for every input scene the execution of both processes either fails or the environment inputs and environment states in output scenes are equal for all interpretations.

We will denote the equivalence of P_1 and P_2 as $P_1 \sim P_2$.

The functional equivalence problem for the case when there are no commutative objects in a given environment was considered in [9] for a more simplified model. It was shown, that the problem could be reduced to the equivalence problem of multidimensional multitape automata, which is solvable [8]. It will be shown below, that the result also holds when the introduced extension is considered.

The equivalence problem in environments with commutative objects will be also considered below.

An equivalent representation of a process, named *sequential execution scheme of the process* - SESP, is defined next. This representation is more convenient for further considerations.

Let $P = \langle \Theta, \Sigma, B, \Gamma, b_s \rangle$ be a process and $\Sigma = \{\sigma_1, \dots, \sigma_\eta\}$. Let also $T_i = \{t_{0_i} + n * \Delta t_i | n \in \{0, 1, 2, \dots\}\}$ is a time constraint for a given situation σ_i of Σ. Let Δt be the least common multiple for all Δt_i, t_0 be the maximum of all t_{0_i}.

The set of states of the SESP(P) corresponds to the set of execution states of the execution algorithm (taking into account the periodicity of time constraints). It is defined as $2^\Sigma \times R \times T \times \{EnvInput\}$, where 2^Σ is the set of pending situations, $R = \{[(\sigma_1, \tau_1), \dots, (\sigma_b, \tau_b)] | \sigma_i.object \neq \sigma_j.object,\ 0 < \tau_i < duration(operation(\sigma_i))\}$

is the set of running situations (τ_i is the time remaining for the completion of operation(σ_i)), $T = \{0, \ldots, t_0 + \Delta t - 1\}$ is the set of possible values of current time, $\{EnvInput\}$ is the set of current environment inputs. Let $NextT(t) = t + 1$ if $t < t_0 + \Delta t - 1$, $NextT(t) = t_0$ if $t = t_0 + \Delta t - 1$. A transition is defined from state $s^{(1)} = (P^{(1)}, R^{(1)}, t^{(1)}, IE^{(1)})$ to state $s^{(2)} = (P^{(2)}, R^{(2)}, t^{(2)}, IE^{(2)})$, where $P^{(1)}, P^{(2)} \in 2^{\Sigma}, R^{(1)}, R^{(2)} \in R, t^{(1)}, t^{(2)} \in T, IE^{(1)}, IE^{(2)} \in \{EnvInput\}$, if $t^{(2)} = NextT(t^{(1)})$ and $s^{(2)}$ can be reached from $s^{(1)}$ by one step of the execution algorithm for $t = t^{(1)}$ for some interpretation.

The execution algorithm is modified to work with SESP in the following way. As the sets $Pending$, $Running$ and $CurrentEnvInput$ are already encoded in the states of SESP, we just need to start from the state corresponding to the input scene and go to a next state corresponding to the $Pending$, $Running$ and $CurrentEnvInput$ sets of the original algorithm, executing the operations and changing the states of objects as in the original algorithm.

Let $StateT(t) = t$ if $t < t_0$, $StateT(t) = t_0 + (t - t_0) mod \Delta t$ otherwise. Let $InputState(S_I) = (b_s, \emptyset, StateT(S_I.t), S_I.EnvInput)$ be the state corresponding to the input scene $S_I, OutputState(S_O) = (\emptyset, \emptyset, StateT(S_O.t), S_O.EnvInput)$ be the state corresponding to the output scene S_O.

The following lemma states the correspondence between a given process and its SESP.

Lemma 1 [9]. *For every interpretation I and every input scene S_I:*

a) if the execution of the process completes successfully with an output scene S_O, there is a path in the SESP from $IS = InputState(S_I)$ to $OS = OutputState(S_O)$, and the execution of the SESP with the input scene S_I reaches OS and vice versa;

b) if the execution of the process fails, the execution of the SESP never completes and vice versa.

3. Equivalence of Processes In Environments With Non-Commutative Objects. A multidimensional multitape automaton modeling a given process P is built below.

The corresponding automaton $A(P)$ has:

- one tape with an alphabet $\{0, 1\}$ for each condition, the dimension of which is the same as the number of objects that the condition uses (condition tapes);
- one 1-dimensional tape for each object for encoding the operation history; it will store operation and message pairs (operation tapes);
- one 1-dimensional tape for reading the start time, input and output messages from (I/O tape);

- one 1-dimensional tape for each operation op, for encoding the function f_{op}; it will store elements from the set $\{Reaction\}$ (message tapes).

The set of states of the automaton $A(P)$ consists of three subsets. These are a set of initialization states, which are used for reading the input scene time and messages, a set of states that are used to read the the output scene messages and a set of main states, used for modeling the execution of the process.

We will assume that the graph of main states of the automaton $A(P)$ is divided into subgraphs - blocks that correspond to the states of the SESP. The block corresponding to the state s is denoted $B(s)$. Each block has one start state, and the only transitions possible between blocks are to a start state. If there is a transition from state s to state s' in the SESP, then there is a transition from block $B(s)$ to block $B(s')$. Below the actions performed in each block are described.

The value of the condition is read from the corresponding condition tape (the heads on condition tapes are not advanced at this point). The output messages (reactions) are read from the corresponding message tapes. Upon completion of an operation, each head on the condition tapes which use the active object (the object, an operation of which just completed) is advanced in the direction corresponding to that object, and the corresponding operation-message pair is read from an operation tape.

The automaton starts by reading the input scene and getting to the corresponding block. After successful execution (the automaton gets to a block corresponding to the end situation batch b_e), the automaton reads the output scene messages from the I/O tape and compares them to the current messages. If they match, then the tapes are accepted, otherwise, they are rejected.

Lemma 2. *The positions of heads of the automaton A on the condition and message tapes are uniquely determined by the positions of heads on operation tapes.*

Let $y = ((y_{11}, y_{12}, \dots), \dots, (y_{n1}, y_{n2}, \dots))$ be the sequences of operation-message pairs of all objects. We will say, that a filling of the tapes models an interpretation, bounded by y, if these operation-message pairs are written on the operation tapes, the message functions are written on the message tapes and the values of the cells of condition tapes, corresponding to any subsequences of y, equal the values of conditions for that interpretation after performing the operations on the given messages. Let I be an interpretation, S_I be an input scene, $y = ((y_{11}, y_{12}, \dots), \dots, (y_{n1}, y_{n2}, \dots))$ be the sequences of operation-message pairs, that the process would perform for I and S_I.

Lemma 3. *For the automaton $A(P)$ working on a filling of tapes modeling I and bounded by y, if in the i^{th} step of the execution of the SESP s is the active state, then in the sequence of active blocks of A the i^{th} would be $B(s)$ and vice*

versa.

Lemma 4. *If a filling of the tapes is such, that there is no interpretation for which the filling is modeling, bounded by some sequences of operation-message pairs, then for any process P corresponding to the signature of tapes, the corresponding automaton A will never stop on the filling.*

Let P_1 and P_2 be two processes, $A(P_1)$ and $A(P_2)$ be the corresponding multi-dimensional multitape automata, constructed in the above-mentioned way.

Theorem 1. $P_1 \sim P_2 \Leftrightarrow A(P_1) \sim A(P_2)$.

Proof. $A(P_1) \sim A(P_2) \Rightarrow P_1 \sim P_2$. It follows from Lemma 3.

$P_1 \sim P_2 \Rightarrow A(P_1) \sim A(P_2)$. Let $P_1 \sim P_2$, and μ is a filling of the tapes on which $A(P_1)$ stops. We'll show that $A(P_2)$ also stops on that filling and the positions of the heads are the same.

According to Lemma 4, there is an interpretation I_μ, for which μ is modeling, bounded by the sequences of operation-message pairs of μ. From $P_1 \sim P_2$ will follow that P_2 gives the same results as P_1 for all input scenes for I_μ, hence $A(P_2)$ stops for all I/O tape fillings (and μ) as $A(P_1)$. The same positions of heads of the two automata follow from Lemma 2.

4. SESP Modeling Via Regular Expressions Over A Partially Commutative Alphabet. Let Y be a finite alphabet, $Y_1, \dots Y_m$ be the partition of the alphabet Y on disjoint non-commutative subsets by a relation ρ. Y is named a partially commutative alphabet [11]. Let R_1 and R_2 be regular expressions in the alphabet Y. If for every word $w \in R_1$ there exists a word $w' \in R_2$, that coincides to w within commutation of symbols from different subsets of the alphabet Y and, vice versa, if for every word $w' \in R_2$ there exists a word $w \in R_1$, that coincides to w within commutation of symbols from different subsets of the alphabet X then regular expressions R_1 and R_2 are named ρ-equivalent and denoted by $R_1 \sim R_2(\rho)$ [11].

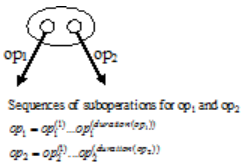
The set of all operations of all objects of an environment Θ will be denoted by $\Theta_{op} = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} o_i.op_j$. Let Y be a partially commutative alphabet [11], $Y = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \{op_{ij}^1 \dots op_{ij}^{duration(o_i.op_j)}\}$ where $op_{ij}^1 \dots op_{ij}^{(duration(o_i.op_j))}$ is a representation of a given operation j of object i as a sequence of sub-operations that are executed within one time unit of the operation duration. Evidently, if $o_i^{(1)}.op_j^{(1)} * o_i^{(2)}.op_j^{(2)} = o_i^{(2)}.op_j^{(2)} * o_i^{(1)}.op_j^{(1)}$ then $op_{i_1j_1}^{(k)} op_{i_1j_1}^{(l)} = op_{i_1j_1}^{(l)} op_{i_1j_1}^{(k)}$.

It is easy to transform a given SESP into a regular expression over the alphabet Y: it is requested just to add for each node transitions for missed operations leading to the endless loop. For a given SESP P the corresponding regular expression will be denoted $R(P)$. The figure below demonstrates the transformation from a source process to an automaton that recognizes the corresponding regular expression.

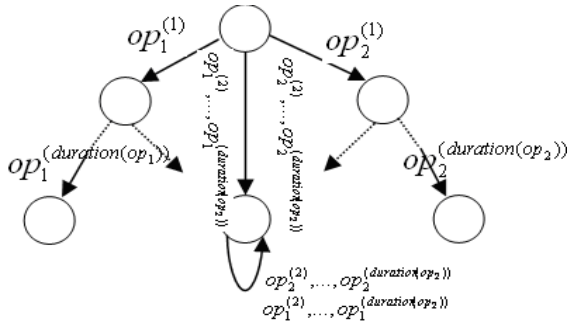
Lemma 5. *For any SESP P_1 and P_2 in the environment Θ with commutative*

objects $P_1 \sim P_2 \Leftrightarrow R_1(P_1) \sim R_2(P_2)(\rho)$.

Proof. $P_1 \sim P_2 \Rightarrow R_1(P_1) \sim R_2(P_2)(\rho)$. Suppose the contrary: processes P_1 and P_2 are equivalent, but the corresponding regular expressions $R_1(P_1)$ and $R_2(P_2)$ are not. The latter, according to the definition, means that there either exists a word w_1 from $R_1(P_1)$ that does not coincide, up to commutation of symbols from different subsets of the alphabet, with any other word w_2 from $R_2(P_2)$ or vice versa. Let's assume that the first case takes place.



Fragment of a source process P corresponding to regular expression $R(P)$



Fragment of the automaton

This, in its turn, according to the transformation of the source process to an automaton, means that there exists at least one sequence of operations of process P_1, that corresponds to the word w_1 from $R_1(P_1)$, which does not have corresponding, in terms of affecting the environment, sequence of operations in process P_2. This contradicts to the assumption that processes P_1 and P_2 are equivalent.

As $R_1(P_1) \sim R_2(P_2)(\rho) \Rightarrow P_1 \sim P_2$ is evidently true, due to the transformation algorithm depicted above, the lemma is proved.

5. Equivalence of Processes In Environments With Commutative Objects.

Theorem 2. *If there exist two commutative objects in an environment* Θ *with more than one operation then the equivalence problem of processes in the environment is unsolvable.*

This can be proved using the technique similar to [10], i.e. via reducing the considered problem to the equivalence problem of non-deterministic multitape automata.

Theorem 3. *The problem of functional equivalence of processes in an environment with commutative objects, when there is only one object with more than one operation, is solvable.*

Per Lemma 5, to prove this theorem, it is sufficient to show that the problem of equivalence of regular expressions over a partially commutative alphabet is

solvable. This is proved in [11]. For the maturity of consideration, the main idea of the proof along with necessary definitions is adduced below.

Let r be a positive integer, $N = \{0, 1, \dots\}$. The set N^r is called an r-dimensional tape. Any element of N^r - $(a_1, \dots, a_r)$ is called a cell of the tape and the numbers $a_1, \dots, a_r$ are called the coordinates of the corresponding cell. The cell $(0, \dots, 0)$ is the initial cell. Let X be a finite alphabet. Any mapping $N^r \to X$ is called a fill of the tape with the symbols of X.

Let $A =< Y, S, \delta, F, s_0 >$ be a deterministic automaton, that recognizes exactly the regular expression R in the input alphabet Y, with the set of states S, the transition function δ, the set of final states F and the initial state s_0. Let $Y = \bigcup_{i=1}^{n} Y_i$, where $|Y_1| \geq 2$, $|Y_i| = 1$, $i = 2, \dots, n$. $|Y_1|$ is denoted by q. Consider $(n + q - 1)$-dimensional tape N^{n+q-1}. q dimensions of the tape are used for expressing the movement on symbols from Y_1, and the rest $n-1$ dimensions are used for expressing the movement on symbols from Y_i, $i = 2, \dots, n$.

Binary representation of cell coordinates, adduced in [7] is used. The binary representation of the initial cell is $(0, \dots, 0)$. The binary representation of any other cell is built basing on the binary representation of the predecessor of the given cell as each cell has only one predecessor. The length of the code of a coordinate of a cell is either equal or greater by one than the length of code of the corresponding coordinate of the predecessor.

The cell $a_1 = (\alpha_{11}, \dots, \alpha_{1n+q-1})$ is called the predecessor of the cell $a_2 = (\alpha_{21}, \dots, \alpha_{2n+q-1})$, and the corresponding predicate, denoted by $\pi(a_1, a_2)$, has the value true, if and only if, $\exists j \in \{1, \dots, n+q-1\}$, $k = 1, \dots, j-1, j+1, \dots, n+q-1$, that

1) $\alpha_{2k} = \alpha_{1k}$.

2) $\alpha_{2j} = \alpha_{1j} + (L + 1)$, if $j = k$, $L = \alpha_{11} + \ldots + \alpha_{1n+q-1}$.

If $\pi(a_1, a_2)$ is true, and the cells a_1 and a_2 differ by the coordinate j, then this is represented by the new predicate π_j as $\pi_j(a_1, a_2)$ is true.

The partially mapping $\varphi : N^n \to 2^S$ is called the set of all execution traces of the automaton A, if and only if:

1) $\varphi_A(0, \dots, 0) = \{s_0\}$, $P(0, \dots, 0) = \emptyset$

2) $\forall a \in N^n \setminus \{0, \dots, 0\}$, $P(a) = \{a^{(j)}_{pred} | \pi_j(a^{(j)}_{pred}, a)$ is true, $j = n_1, \dots, n_k$, $k \leq n + q - 1\}$, $\varphi_A(a) = \Delta(\varphi_A(a^{(n_1)}_{pred}), y_{n_1}) \cup \ldots \cup \Delta(\varphi_A(a^{(n_k)}_{pred}), y_{n_k})$.

The finite subset of execution traces, where the sum of coordinates of each cell is less or equal to, $k - 1$, will be called an (execution) trace word of length k. The set of all trace words is denoted by Ω_A. The set of all cells, used in the given trace word ω is denoted by U_ω.

The part of a trace word ω, where sum of coordinates of each cell is equal to k, is called the k^{th} diagonal of the trace word ω and denoted by $d_k(\omega)$. The length

of $d_k(\omega)$ is equal to $k+1$. The length $l(\omega)$ of a trace word ω is equal to the number of the diagonals it contains.

For a given trace word ω a path $p = a_{p_1} \dots a_{p_m}$, $m \geq 1$, $a_{p_j} \in U_\omega$, $j \in \{1, \dots, m\}$ is defined as sequence of cells where $\pi(a_{p_v}, a_{p_{v+1}})$, $v = 1, \dots, m-1$.

For a given path $p = a_{p_1} \dots a_{p_m}$ the word $\chi_p = y_{p_1} \dots y_{p_{m-1}}$ in the alphabet Y is called the characteristics of the path p if and only if, $\forall j \in \{1, \dots, m\}$, $i \in \{1, \dots, n+q-1\}$:

1) $y_{p_j} \in Y_i$, $i \neq 1 \Rightarrow \alpha_{p_j+1}[i] = a_{p_j+1}[i] + 1$,

2) $y_{p_j} \in Y_1 \Rightarrow \alpha_{p_j+1}[i] = \alpha_{p_j}[i] + (L+1)$,

$L = \alpha_{p_j}[1] + \ldots + \alpha_{p_j}[n+q-1]$.

The path $p = a_{p_1} \dots a_{p_m}$ is called complete, if $a_{p_1} = (0, \dots, 0)$. A complete path $p = a_{p_1} \dots a_{p_m}$ is called accepted by the automaton A, if $\varphi_A(a_{pm})$ contains final state of automaton A. The coordinates of cell a_{p_m} are called canonical form of the complete path $p = a_{p_1} \dots a_{p_m}$.

For each automaton A, the set of all accepted paths in a trace word ω is denoted by $AP_A(\omega)$, and the set of their canonical forms - $CF_{AP_A(\omega)}$.

Suppose that automata A_1 and A_2 recognize exactly regular expressions R_1 and R_2, correspondingly, and S_1, S_2 - are their sets of states. Also suppose that $k = 2^s - 1$, where $S = |S_1| + |S_2|$, ω_1 and ω_2 are trace words of automata A_1 and A_2 of length k.

Lemma 6. $CF_{AP_{A_1}(\omega_1)} = CF_{AP_{A_2}(\omega_2)} \Leftrightarrow R_1 \sim R_2(\rho)$.

Assume R_1 is not ρ-equivalent to R_2, and there are no canonical forms of paths of the length less or equal to k in trace words ω_1 and ω_2 that are recognized by the automaton A_1, but are not recognized by the automaton A_2, or, vice versa, are recognized by the automaton A_2, but are not recognized by the automaton A_1.

Due to assumption there is a canonical form of corresponding complete paths $p^{(j)} = a_{p_1} \dots a_{p_m}$ with a length greater than k, $m > k$, which is recognized by one of the automata, say, A_1, but is not recognized by the automaton A_2. This means that $\varphi_{A_1}(a_{p_m})$ contains final state of automaton A_1, whereas $\varphi_{A_2}(a_{p_m})$ does not contain a final state of the automaton A_2. As $m > k$, there exist two cells $a^{(j)}$ and $a^{(j')}$ in the path p, such that the sum of coordinates of the cell $a^{(j')}$ is greater than the sum of coordinates of the cell $a^{(j)}$, $\varphi_{A_j}(a^{(j)}) = \varphi_{A_j}(a^{(j')})$, $j = 1, 2$, and $p^{(j)} = a_{p_1} \dots a^{(j)} \dots a^{(j')} \dots a_{p_m}$.

Let $\chi_p^{(j')}$ be the characteristics of the sub-path $p^{(j')} = a^{(j')} \dots a_{p_m}$ of the path p, and $p^{(j'')} = a^{(j)} \dots a''$ is a path starting from the cell a, to some cell a'', which has the same characteristics $\chi_p^{(j')}$. It is evident that such a path exists. The sub-path of the path p, which starts from a_{p_1} and ends with the predecessor of the cell $a^{(j)}$ is denoted by $p^{(j''')}$. The concatenation of two paths $p^{(j''')}$ и $p^{(j'')}$ is denoted by $p_{new}^{(j)} = p^{(j''')}p^{(j'')}$. It's length is less than the length of the initial path $p^{(j)}$. Evidently,

$\varphi_{A_1}(a'') = \varphi_{A_1}(a_{p_m})$ contains the final state of the automaton A_1, and $\varphi_{A_2}(a'') = \varphi_{A_2}(a_{p_m})$ does not contain the final state of the automaton A_2. If the length of the path p_{new} is still greater than k, similar considerations shall be performed until the obtained path has a length not exceeding k. But this contradicts to the initial suggestion that if a path is accepted in trace words ω_j by the automaton A_j Then there exists another path with the same canonical form which is accepted by the automaton $A_{j'}$ $j, j' \in \{1, 2\}$, $j \neq j'$.

As the assumption $R_1 \sim R_2(\rho) \Rightarrow CF_{AP_{A_1}(\omega_1)} = CF_{AP_{A_2}(\omega_2)}$ is obviously true, lemma is proved.

Yerevan State University
Institute for Informatics and Automation Problems NAS RA

H. A. Grigoryan, A. S. Shoukourian

Equivalence of Processes in Object-Oriented Environments with Commutative and Non-Commutative Objects

The equivalence of processes is an important constituent of process optimization. Functional equivalence problem for processes in object-oriented environments is considered. Two cases are investigated: when the turn of executing operations for objects is essential and when for some objects it is not essential. In the latter case the objects are named commutative. It is shown that the equivalence problem for environments that contain two or more commutative objects with more than one operation is insolvable. It is also shown that if there is only one object with two or more operations among commutative objects then the equivalence problem is solvable.

Հ. Ա. Գրիգորյան, Ա. Ս. Շուքուրյան

Պրոցեսների համարժեքությունը կոմուտատիվ եւ ոչ կոմուտատիվ օբյեկտներով օբյեկտակողմնորոշված միջավայրերում

Պրոցեսների համարժեքությունը դրանց օպտիմիզացիայի կարեւոր բաղկացուցիչ է: Այս հոդվածում դիտարկված է պրոցեսների ֆունկցիոնալ համարժեքության խնդիրը օբյեկտակողմնորոշված միջավայրերում: Ուսումնասիրված են երկու դեպք՝ երբ օբյեկտների համար գործողությունների կատարման հաջորդականությունը կարեւոր է, եւ երբ, ինչ-որ օբյեկտների համար, այն կարեւոր չէ: Վերջին դեպքում օբյեկտները կոչվում են կոմուտատիվ: Ցույց է տրված, որ պրոցեսների համարժեքության խնդիրը լուծելի չէ այն դեպքում, երբ միջավայրում գոյություն ունեն երկու կամ ավելի կոմուտատիվ օբյեկտներ, որոնք ունեն մեկից ավելի գործողություն: Ցույց է տրված նաեւ, որ եթե կոմուտատիվ օբյեկտների մեջ գոյություն ունի միայն մեկ օբյեկտ երկու կամ ավել գործողություններով, ապա համարժեքության խնդիրը լուծելի է:

Г. А. Григорян, А. С. Шукурян

Эквивалентность процессов в объектно-ориентированных средах с коммутативными и некоммутативными объектами

Распознавание эквивалентности процессов является важной составляющей их оптимизации. В статье рассмотрена проблема функциональной эквивалентности процессов в объектно-ориентированной среде. Исследованы два следующих случая: когда очередность выполнения операций для объектов важна, и когда для некоторых объектов она не имеет значения. В последнем случае объекты называются коммутативными. Показано, что проблема эквивалентности в средах, содержащих два или более коммутативных объекта с более чем одной операцией, неразрешима. Также показано, что если среди коммутативных объектов существует лишь один объект с двумя или более операциями, то проблема эквивалентности разрешима.

References

1. *Bergstra J.A.* In: A. Ponse and S. Smolka (eds.). Handbook of Process Algebra. Elsevier Science. 2001.

2. *Baeten J.C.M.* - Theoretical Computer Science. 2005. V. 335. Issue 2-3. P. 131-146.

3. *Raulefs P.* - IFIP World Computer Congress '94. 1994. V. 2. P. 18-30.

4. *Bard Bloom* - Formal Asp. Comput. 1994. V. 6(3). P. 317-338.

5. *Magnani L., Nersessian N.J., Thagard P.* Model-Based Reasoning in Scientific Discovery. Springer. 1999.

6. *Vargas-Villamil F. D., Rivera D.E., Kempf K.G.* - IEEE Transactions on Control Systems Technology. 2003. V. 11. N 4. P. 578-87.

7. *Годлевский А. Б., Летичевский А. А., Шукурян С. К.* - Кибернетика. 1980. N 6. С. 1-7. (*A. B. Godlevskii, A. A. Letichevskii, S. K. Shukuryan* Cybernetics and Systems Analysis. 1980. N. 6. P. 1-7.)

8. *Grigorian H., Shoukourian S.* - Journal of Computer and System Sciences. 2008. V. 74. Issue 7. P. 1131-1138.

9. *Grigoryan H. A.* - Reports of the National Academy of Sciences of Armenia. 2008. V. 108. N 1. P. 50-59.

10. *Тузов В. А.* - Кибернетика. 1971. N5. С. 28-32. (*V.A.Tuzov* - Cybernetics and Systems Analysis. 1971. V. 7. N. 5. P. 778-789.)

11. *Шукурян А. С.* - Кибернетика и системный анализ. 2009. N3. С. 3-11. (*A.S. Shoukourian* - Cybernetics and Systems Analysis. 2009. V. 45. N. 3. P. 387-396.)