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

Том 108 2008 № 1

MATHEMATICS

УДК 519.681

H. A. Grigoryan

Equivalence of Processes in an Object-Oriented Environment

(Submitted by academician S.K. Shoukourian 18/I 2008)

Keywords: *process, equivalence, multitape automata, multidimensional automata, object-oriented environment*

1. Introduction. The equivalence of processes is important for the process optimization. Among various models of processes [1] a model is chosen which is defined over an object-oriented environment [2]. The article considers the problem of process equivalence within the mentioned model. Some items of the model, which were not concisely defined before, are formalized and formal semantics of processes is adduced for the considered case. Some restrictions to timing, which do not violate the idea of Virtual Factory and are coordinated with the author of the model, are introduced. The problem of process equivalence is formulated and it is shown that this problem can be reduced to the equivalence problem of automata with multidimensional tapes, where the motion of the heads is monotone (no backward motion) in all directions (multidimensional multitape automata - MMA) [3]. In its turn the latter problem is solvable [4,5].

2. Model Definition. The following model equations are defined:

$$\begin{aligned} Object &= (ID, \{State\}, \{Operation\}) \\ MSG &= \{msg_1, \ldots, msg_k\} \\ Operation &= \{State\} \times MSG \to \{State\} \times 2^{MSG} \\ Duration &= \{Operation\} \to \{1, 2, \ldots\} \\ Environment &= \{Object\} \end{aligned}$$

The state sets and operation sets of different objects are considered disjoint. Environments containing finite set of objects are considered further. Let $\Theta =$

$\{o_1, \ldots, o_n\}$ be an environment.

$$
\begin{aligned}
EnvState &= (o_1.State, \ldots, o_n.State) \\
EnvStateSet &= o_1.StateSet \times \ldots \times o_n.StateSet \\
TimeConstraint &= \{t_0 + n^* \Delta t | n \in \{0, 1, 2, \ldots\}\} \text{ where } t_0 \text{ and } \Delta t \text{ are non-negative integers} \\
BasicPred &= \{\pi_1, \ldots, \pi_r\}, \text{ where } \pi_j \text{ is a predicate symbol of arity } n_j \\
Condition &= \pi_j(o_{i_1}.State, \ldots, o_{i_{n_j}}.State), \text{ where } \pi_j \in BasicPred \\
Assertion &= (Condition, TimeConstraint) \\
Situation &= (Object, Assertion) \\
SituationBatch &= \{Situation\} \\
Scene &= (t, EnvState, 2^{MSG})
\end{aligned}
$$

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 $b \in B$, $\omega \in \sigma.Object.Operations$,
b_s - is the start situation batch.
We will say that an *interpretation* is defined for a given process, if the following is defined:

- initial $EnvState : es_0 = (st_0^{(1)}, \ldots, st_0^{(n)})$
- functions $f_{op} : \{o_i.State\} \times MSG \rightarrow \{o_i.StateSet\} \times 2^{MSG}$ for all operation symbols op of all objects, $op \in o_i.Operation$
- functions $\rho_{\pi_j} : o_{i_1}.StateSet \times \ldots \times o_{i_{n_j}}.StateSet \rightarrow \{TRUE, FALSE\}$ (n_j is the arity of the corresponding predicate symbol) for all predicate symbols.

3. Process Semantics. The semantics of a process interpretation is defined by 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.

The 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.
- **CurrentMessages** - a set of currently available messages.
- **nextNode(σ)**= b, if $(\sigma, b, \omega) \in \Gamma$.
- **operation(σ)**= ω, if $(\sigma, b, \omega) \in \Gamma$.
- **receivedMsg(σ)**= $TRUE$, if a message needed for operation(σ) is in **CurrentMessages**.
- **canExecute(σ)=receivedMsg(σ)** & σ.Assertion.Condition, where σ is a situation.
- **beginExecution(σ)** - removes the message needed for operation(σ) from **CurrentMessages**.
- **finalizeExecution(σ)** - changes the state of the corresponding object and adds the output messages of operation(σ) to **CurrentMessages**.
- **executed(σ)**= $TRUE$, if duration(operation(σ)) 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(σ, cond(σ))** - iterates sequentially over any situation σ for which a given condition over σ - **cond(σ)** is true, according to the order defined in Σ.

Execution Algorithm

```
Pending = Running = ∅;
CurrentMessages = InputScene.Messages;
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, $\sigma \in$Pending & canExecute(σ) &
$t \in \sigma$.Assertion.TimeConstraint &
($\forall\sigma_1 \in$Running σ_1.object$\neq \sigma$.object))
begin
move(σ, Pending, Running);
beginExecution(σ);
end

if(Running= $\emptyset$ & ($\forall\sigma \in$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, CurrentMessages) will be called an *execution state* of the algorithm.

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

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

It will be shown, that in this case the equivalence problem of processes can be reduced to the equivalence problem of multidimensional multitape automata (MMA) and, thus, is solvable [4, 5]. The reduction will be done in two steps: first - an execution scheme will be constructed for the process, second - the corresponding MMA will be built.

5. Sequential Execution Schemes of Processes. An equivalent representation of a process (named *sequential execution scheme of the process* - SESP) will be constructed first. This representation is more convenient for further considerations.

Let $P = \langle \Theta, \Sigma, B, \Gamma, b_s \rangle$ be a process and $\Sigma = \{\sigma_1, \ldots, \sigma_\eta\}$. Let also $T_i = \{t_{0i} + n^* \Delta t_i | n \in \{0, 1, 2, \ldots\}\}$ 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_{0i}.

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 $2^\Sigma \times R \times T \times 2^{MSG}$, where 2^Σ is the set of pending situations, $R = \{|(\sigma_1, \tau_1), \ldots, (\sigma_b, \tau_b)|| \sigma_i.object \neq \sigma_j.object,\ 0 < \tau_i < duration(operation(\sigma_i))\}$ is the set of running situations (τ_i is the time remaining for the completion of $operation(\sigma_i)$).

$T = \{0, \ldots, t_0 + \Delta t - 1\}$ is the set of possible values of current time, 2^{MSG} is the set of current messages. 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 $s^{(1)} = (P^{(1)}, R^{(1)}, t^{(1)}, M^{(1)})$ to state $s^{(2)} = (P^{(2)}, R^{(2)}, t^{(2)}, M^{(2)})$, where $P^{(1)}, P^{(2)} \in 2^{\Sigma}$, $R^{(1)}, R^{(2)} \in R$, $t^{(1)}, t^{(2)} \in T$, $M^{(1)}, M^{(2)} \in 2^{MSG}$, 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.

For example, the fragment of the SESP for the start fragment of a process in Fig. 1a is shown in Fig. 1b. In the example $MSG = \{a\}$, $duration(x) = duration(y) = 2$, $s_1 > s_2$, $t + 1 = NextT(t)$. F=FALSE, T=TRUE.

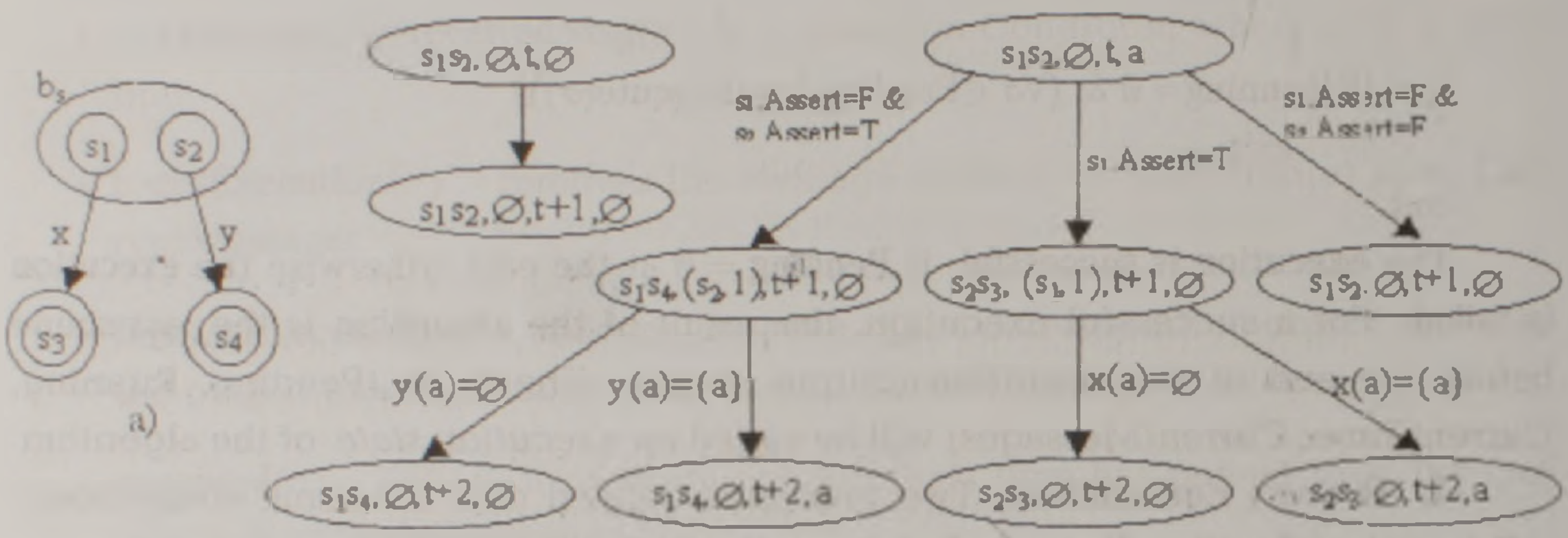

Fig. 1.

The execution algorithm is modified to work with SESP in the following way. As the sets *Pending*, *Running* and *CurrentMessages* 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 *CurrentMessages* 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 $InputState(S_I) = (b_s, \emptyset, StateT(S_I.t), S_I.Messages)$ be the state corresponding to the input scene S_I, $OutputState(S_O) = (\emptyset, \emptyset, StateT(S_O.t), S_O.Messages)$ be the state corresponding to the output scene S_O.

Execution Algorithm for SESP

```
CurrentState = InputState(InputScene);

for(;;)
begin
```

```
//the execution is at state CurrentState

    foreach(σ, (σ, τ) ∈CurrentState.Running)
    if(τ = 1)
        begin
            finalizeExecution(σ);
            remove(CurrentState.Running, (σ, τ));
            add(CurrentState.Pending, nextNode(σ));
        end
    else
        − − τ;

    foreach(σ, σ ∈CurrentState.Pending & canExecute(σ) &
        CurrentState.t ∈ σ.Assertion.TimeConstraint &
        (∀(σ1, τ1) ∈CurrentState.Running σ1.object≠ σ.object))
    begin
        remove(CurrentState.Pending, σ);
        add(CurrentState.Running,
                (σ, duration(operation(σ))-1));
        beginExecution(operation(σ));
    end

        if(CurrentState.Running= ∅ & CurrentState.Pending= ∅)
            exit;

        CurrentState.t = NextT(CurrentState.t)
    end
```

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

Lemma 1. *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.

Proof. The proof is by induction on the number of steps taken. The SESP state $InputState(S_I)$ corresponds to the initial execution state of the execution algorithm with an input scene S_I, and the transitions result in corresponding states, as can be seen from the two algorithms. The final SESP states $OutputState(S_O)$

have $Running = \emptyset$ & $Pending = \emptyset$, so correspond to successful execution. In case of failure of the execution of the process, the execution of the SESP will be reduced just to a loop between states, which differ only by time.

6. Reduction of the Equivalence Problem of Processes to the Equivalence Problem of MMA. The definition of MMA from [3, 5] is adduced here for a convenience of reading. Let d be a positive integer, $N = \{0, 1, \dots\}$. The set N^d is called a **d-dimensional tape**. Any element of N^d - $(a_1, \dots, a_d)$ is called a cell of the tape and the numbers $a_1, \dots, a_d$ are called the coordinates of the corresponding cell. The cell $(0, \dots, 0)$ is the initial cell. Let X be a finite alphabet. Mappings $N^d \to X$ are called fills of the tape with the symbols of X.

The set $S = \{(n_1, m_1), \dots, (n_h, m_h)\}$, where n_i, m_i $(1 \leq i \leq h)$ are natural numbers and for all $1 \leq i, j \leq h$, $n_i = n_j \Leftrightarrow i = j$, is called a **signature** of the MMA. The signature defines the quantity and arity of the tapes - if $(i, j) \in S$, then the automaton with a signature S has exactly j i-dimensional tapes.

$A =< Q = Q_1 \cup \dots \cup Q_m, X, q_0, Q_F, \varphi, \psi >$ (Q - set of states (Q_i contains those states in which the ith tape is being read, $Q_i \cap Q_j = \emptyset$, if $i \neq j$), X - input alphabet, q_0 - initial state, Q_F - final states, $\varphi : Q \times X \to Q$ - transition function, $\psi : Q \times X \to \{1, \dots, c\}$ - movement direction function) will be called a multidimensional multitape automaton (MMA) with signature S.

The filled part of the d-dimensional tape, the sum of coordinates of each cell is less than or equal to $i - 1$, will be called a **d-dimensional word** of length i. The execution of the automaton will be considered on words of finite length. The m-tuple of multidimensional words $(p_1, \dots, p_m)$ will be called an **m-tape word** with a signature S, if the number of u-dimensional words is v, and $(u, v) \in S$. A word is accepted if the automaton is in a final state after reading the entire word. If an automaton A (with any signature) accepts / doesn't accept the word w, it will be denoted as $A(w) = 1$ / $A(w) = 0$, correspondingly.

A_1 and A_2 multidimensional multitape automata will be called **equivalent**, if for every word w $A_1(w) = A_2(w)$, and the positions (coordinates) of the heads on all tapes are the same, if $A_1(w) = A_2(w) = 1$. The equivalence of two automata will be denoted $A_1 \sim A_2$.

An MMA modeling a given process P will be described below.

The corresponding automaton A will have:

- 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 2^{MSG} (the output message sets) (message tapes).

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

We will consider the set of main states of the automaton A to be partitioned into blocks that correspond to the states of the SESP. The block corresponding to the state s will be denoted $B(s)$. Each block has one start state, and the only transitions possible between blocks are to a start state. There is a transition from block $B(s)$ to block $B(s')$, if there is a transition from state s to state s' in the SESP. Below we will describe the actions in each block.

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 message sets 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.*

Proof. The positions of the heads on the condition and message tapes depend only on the number of operations performed by each object, so the lemma is true.

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 that opera-