To The Parallel Composition of Timed Finite State Machines

Olga Kondratyeva, Maxim Gromov
Radiophysics Faculty
Tomsk State University
Tomsk, Russia
kondratyeva.olga.vic@gmail.com, gromov@sibmail.com

***Abstract*—This paper deals with the problem of the parallel composition construction for two Timed Finite State Machines (TFSMs). As a key to the solution of this problem we use parallel composition of common Finite State Machines (FSMs). We transform given TFSMs to FSMs and prove theorem, that obtained FSMs correctly describe behaviour of the given TFSMs. Then we build parallel composition of these FSMs, which being transformed back to TFSM, gives desired parallel composition of the given TFSMs**

Keywords-Finite State Machine; Timed Finite State Machine; parallel composition

I. INTRODUCTION

The Timed Finite State Machine (TFSM) is a model based on well-known Finite State Machine (FSM), which allows explicit description of a time aspects of system behaviour. For example, reaction of a system can be different depending on the time moment an input action is applied to it. In the last few years the interest to the various problems of TFSM has increased. The main lines of researches covered by the post papers are the analysis problems: relations between TFSMs [1, 2] and test generation methods against those relations [3, 4].

In our paper we consider a problem of synthesis, namely the problem of parallel composition construction of two TFSMs. This procedure gives an instrument to build complex systems from simple ones, each described by a TFSM. Also, the approach we used in this paper to describe a parallel composition construction procedure opens the way for solving various problems of TFSMs.

II. PRELIMINARIES

In this section we give some notions and definitions, which we shall use all over the paper.

A. Language

An *alphabet* is a finite non-empty set of symbols and as usual, given an alphabet X, we denote X^* the set of all finite sequences (words) of symbols from X including the empty word ε. The number of symbols in a sequence we shall call *length* of this sequence; by definition, length of the empty word is zero. A subset $L \subseteq X^*$ is a *language* over alphabet X.

Let language L be defined over alphabet Y and X be a non-empty subset of Y. The *X-restriction* $L_{\Downarrow X}$ of the language L is derived by deleting from each sequence of L each symbol of the set $Y \backslash X$. When the language L is defined over alphabet X, and Y is some alphabet that is disjoint with X, consider the mapping $\varphi: X \to 2^{(X \cup Y)^*}$ such, that $\varphi(x) = \{\alpha x \beta : \alpha, \beta \in Y^*\}$. This mapping can be extended over sequences from X^* as follows. Let γ be a sequence from X^* and x be a symbol from X, then $\varphi(\varepsilon) = Y^*$ and $\varphi(x\gamma) = \varphi(x) \cdot \varphi(\gamma)$, where the sign "$\cdot$" stands for concatenation of sequences. We shall call the language $L_{\Uparrow Y} = \{\varphi(\gamma) : \gamma \in L\}$ the *Y-expansion* of language L.

B. Finite automata

There exists a special set of languages which can be described by the use of finite automata; those are regular languages, which are closed under union, concatenation, complementation, intersection and also under restriction and expansion.

A finite automaton (FA) is a 5-tuple $\mathcal{S} = \langle S, A, s_0, \delta_S, Q \rangle$, where S is a non-empty finite set of states with the designated initial state s_0, A is a finite alphabet of actions, $\delta_S \subseteq S \times A \times S$ is a transition relation, and $Q \subseteq S$ is a set of final (accepting) states. If $(s_1, a, s_2) \in \delta_S$, then we say, that automaton $\mathcal{S}$ in the state s_1 takes action a, and changes its state to the state s_2; the state s_2 is called an *a-successor* of the state s_1 and we denote by $suc_S(s_1, a)$ the set of all a-successors of the state s_1. Function suc_S can be extended over sequences from A^* as follows:

$$suc_S(s_1, \beta a) = \{suc_S(s_2, a) : s_2 \in suc_S(s_1, \alpha)\}.$$

By the definition $suc_S(s_1, \varepsilon) = s_1$.

Finite automaton $\mathcal{S}$ is called *deterministic* if for each pair $(s_1, a) \in S \times A$ there is at most one state $s_2 \in S$ such that $(s_1, a, s_2) \in \delta_S$, i.e. $|suc_S(s_1, a)| \le 1$, otherwise, the finite automaton is *non-deterministic*.

Finite automaton $\mathcal{S}$ is called *complete* if for each pair $(s_1, a) \in S \times A$ there is at least one state $s_2 \in S$ such that $(s_1, a, s_2) \in \delta_S$, i.e. $|suc_S(s_1, a)| \ge 1$, otherwise, the finite automaton is *partial*.

Let us consider a word $\beta \in A^*$. Automaton $\mathcal{S}$ *recognizes* or *accepts* β if there exists an accepting state $q \in Q$ such that q is a β-successor of the initial state, i.e. $q \in suc_S(s_0, \beta)$. The

This paper is partially supported by Russian Foundation for Basic Research (Grant 10-08-92003-HHC_a)

set L_S of all sequences, which are accepted by S, is the *language accepted* by the automaton or simply the *language of the automaton* S. The language of a finite automaton is a regular language [5].

C. Finite State Machines

To describe behaviour of a system, which transforms sequences over one (input) alphabet into sequences over another (output) alphabet, special kind of automata, called Finite State Machine, is usually used [6].

A finite state machine (FSM) is a 5-tuple $S = \langle S, I, O, s_0, \lambda_S \rangle$, where S is a non-empty finite set of states with initial state s_0, I and O are disjoint finite input and output alphabets, $\lambda_S \subseteq S \times I \times O \times S$ is the transition relation. If $(s_1, i, o, s_2) \in \lambda_S$, then we say, that the FSM S in the state s_1 gets the input action i, produces the output action o and changes its state to s_2; the state s_2 is called an *i*/*o-successor* of the state s_1. The set of all *i*/*o*-successors of the state s_1 is denoted $suc_S(s_1, i, o)$, while

$$suc_S(s_1, i) = \{s_2 \in S: \exists\, o \in O \text{ such that } s_2 \in suc_S(s_1, i, o)\}$$

is the set of all *i-successors* of the state s_1.

Functions $suc_S(s_1, i, o)$ and $suc_S(s_1, i)$ can be extended to the sequences $\alpha \in I^*$ and $\beta \in O^*$, where lengths of α and β are equal, as follows:

$$suc_S(s_1, \alpha i, \beta o) = \{suc_S(s_2, i, o): s_2 \in suc_S(s_1, \alpha, \beta)\}$$

and

$$suc_S(s_1, \alpha i) = \{suc_S(s_2, i): q \in suc_S(s_1, \alpha)\}.$$

By the definition $suc_S(s_1, \varepsilon) = suc_S(s_1, \varepsilon, \varepsilon) = s_1$.

FSM S is *deterministic* if for each pair $(s_1, i) \in S \times I$ there is at most one pair $(o, s_2) \in S \times O$ such that $(s_1, i, o, s_2) \in \lambda_S$, i.e. $|suc_S(s_1, i)| \le 1$, otherwise, FSM S is *non-deterministic*.

FSM S is *complete* if for each pair $(s_1, i) \in S \times I$ there is at least one pair $(o, s_2) \in S \times O$ such that $(s_1, i, o, s_2) \in \lambda_S$, i.e. $|suc_S(s_1, i)| \ge 1$, otherwise, FSM S is *partial*.

FSM S is *observable* if for each triple $(s_1, i, o) \in S \times I \times O$ there is at most one state $s_2 \in S$ such that $(s_1, i, o, s_2) \in \lambda_S$, i.e. $|suc_S(s_1, i, o)| \le 1$, otherwise, FSM S is *non-observable*.

A sequence $\sigma = (i_1, o_1)(i_2, o_2)\ldots(i_n, o_n) \in (I \times O)^*$ is called a *trace* of given FSM S if the set of α/β-successors, where $\alpha = i_1 i_2 \ldots i_n$ and $\beta = o_1 o_2 \ldots o_n$, of the initial state of S is non-empty, i.e. $suc_S(s_0, \alpha, \beta) \neq \varnothing$. The set of all traces of the FSM is the *language* L_S of the FSM S. Further, talking about traces of an FSM, we assume that a sequence σ and the corresponding pair α/β are equivalent notions.

Given FSM $S = \langle S, I, O, s_0, \lambda_S \rangle$, the automaton $Aut(S)$ is a 5-tuple $\langle S \cup (S \times I), I \cup O, s_0, \delta_S, S \rangle$, where for each transition (s_1, i, o, s_2) in λ_S there are two transitions $(s_1, i, (s_1, i))$, and $((s_1, i), o, s_2)$ in δ_S. Since the language L_S^{aut} of the automaton $Aut(S)$ is the language of the FSM [7] it holds that $L_S^{aut} \subseteq (IO)^*$, where IO is concatenation of alphabets I and O.

D. Timed Finite State Machines

A *timed finite state machine* (TFSM) is a 6-tuple $S = \langle S, I, O, s_0, \lambda_S, \Delta_S \rangle$, where the 5-tuple $\langle S, I, O, s_0, \lambda_S \rangle$ is an FSM and $\Delta_S: S \to S \times (\mathbb{N} \cup \{\infty\})$ is a time-out function. If $\Delta_S(s_1) = (s_2, n)$, then the TFSM S in the state s_1 will wait an input action for n time units (ticks), and if none arrives it will move to the state s_2 (possibly the same as s_1), without producing any output. If $\Delta_S(s_1) = (s_2, \infty)$, then we require $s_2 = s_1$ and the TFSM can stay in the state s_1 infinitely long, waiting for an input. Definitions of a deterministic, complete and observable TFSM are based on the corresponding definitions for underlying FSM.

A special timed or clock variable can be associated with a TFSM; this variable counts time ticks passed from the moment when the last transition has been executed and is reset to 0 after each transition (input-output or time-out). In this paper, for the sake of simplicity, we assume that the output is produced immediately after a machine gets an input, i.e., we do not consider delays when executing transitions.

A pair $(i, t) \in I \times (\mathbb{N} \cup \{0\})$ is a *timed input* meaning that the input i is submitted to the TFSM t ticks later than the previous output has been produced. A sequence of inputs is a timed input sequence.

We also define a special function $time_S$ [1] as follows:

1. $time_S(s, t) = s$ for all $t \in \mathbb{N} \cup \{0\}$ if $\Delta_S(s) = (s, \infty)$.
2. $time_S(s_1, t) = s_1$ for all $t < T$ and $\Delta_S(s_1) = (s_2, T)$.
3. $time_S(s_1, t) = s_2$ for $t = T$ and $\Delta_S(s_1) = (s_2, T)$.
4. for $t > T$ and $\Delta_S(s_1) = (s_2, T)$ define recursively $time_S(s_1, t) = time_S(s_2, t - T)$, i.e. there is a sequence $s_1, s_2, \ldots s_k$ such that for each $j = 1 \ldots k - 1$ it holds $\Delta_S(s_j) = (s_{j+1}, T_j)$ and $T_1 + T_2 + \ldots + T_{k-1} \le t < T_1 + T_2 + \ldots + T_{k-1} + T_k$, then $time_S(s_1, t) = s_k$.

The function suc_S is defined similar to that defined for an FSM and is extended to timed inputs as follows: $suc_S(s, (i, t), o) = suc_S(time_S(s, t), i, o)$.

A sequence

$$\sigma = (i_1, t_1, o_1)(i_2, t_2, o_2)\ldots(i_n, t_n, o_n) \in [I \times (\mathbb{N} \cup \{0\}) \times O]^*$$

is called a *functional trace* of a TFSM S, if the following holds $suc_S(s_0, \alpha, \beta) \neq \varnothing$, where $\alpha = (i_1, t_1)(i_2, t_2)\ldots(i_n, t_n)$ and $\beta = o_1 o_2 \ldots o_n$. The set L_S of all functional traces of the TFSM S is the *f-language of the TFSM* S. Here we again assume, that the pair α/β and the sequence σ are the equivalent notions, when speaking about f-language of a TFSM.

E. Equivalence of automata, FSMs and TFSMs

Two finite automata S and P with languages L_S and L_P are said to be *equivalent* if $L_S = L_P$.

Two FSMs S and P with languages L_S and L_P are said to be *equivalent* if $L_S = L_P$.

Two TFSMs S and P with f-languages L_S and L_P are said to be *equivalent* if $L_S = L_P$.

III. PARALLEL COMPOSITION

In this paper we propose definition of parallel composition for two TFSMs. This definition relies on the definition of FSM parallel composition and the latter is defined in terms of parallel composition of corresponding automata. For that reason we also describe the conversion procedure [8] of a TFSM into an FSM, which then is used for the parallel composition construction. We also prove, that built FSM correctly reflects the language of a given TFSM.

A. Parallel composition of languages

Given pairwise disjoint alphabets X, Y, Z, languages L_1 over $X \cup Y$ and L_2 over $Y \cup Z$, the *parallel composition* of languages L_1 and L_2 is the language

$$L = [(L_1)\Uparrow_Z \cap (L_2)\Uparrow_X] \Downarrow_{X \cup Z},$$

defined over $X \cup Z$ and denoted $L_1 \lozenge_{X \cup Z} L_2$ or just $L_1 \lozenge L_2$ when the union $X \cup Z$ is clear from context.

B. Parallel composition of automata

Given two finite automata $S = \langle S, I \cup U, s_0, \delta_S, Q_S \rangle$ and $P = \langle P, U \cup O, p_0, \delta_P, Q_P \rangle$, the automaton $C = \langle C, I \cup O, c_0, \delta_C, Q_C \rangle$ is a *parallel composition of automata* S and P, denoted $C = S \lozenge P$, iff $L_C = L_S \lozenge L_P$. To obtain composition of automata define expansion and restriction over automata as follows.

Given disjoint alphabets I and O and an automaton $S = \langle S, I, s_0, \delta_S, Q_S \rangle$. *O-expansion* of S is an automaton $S_{\Uparrow O} = \langle S, I \cup O, s_0, \delta_S \cup \mu_S, Q_S \rangle$, where $\mu_S \subseteq S \times O \times S$ contains all triples (s, o, s), such that $o \in O$ and $s \in S$, i.e. to expand an automaton we add loops marked with all symbols of alphabet O for each state.

Given disjoint alphabets I and O and an automaton $S = \langle S, I \cup O, s_0, \delta_S, Q_S \rangle$. *I-restriction* of S is an automaton $S_{\Downarrow I} = \langle S, I \cup \{\tau\}, s_0, \mu_S, Q_S \rangle$, where for each transition $(s_1, a, s_2) \in \delta_S$ we add transition (s_1, a, s_2) into μ_S in case $a \in I$, while we add transition (s_1, τ, s_2) into μ_S in case $a \in O$, i.e. to restrict an automaton we replace all symbols of alphabet O by special symbol τ. An automaton without τ-moves can be derived by the determinization procedure [9].

Now, the procedure of parallel composition construction of two given automata S and P can be described by the formula:

$$C = [S_{\Uparrow O} \cap P_{\Uparrow I}] \Downarrow_{I \cup O}.$$

C. Parallel composition of FSMs

Following [7], we define the parallel composition of two FSMs (Fig. 1) based on their corresponding automata. However, the language of the parallel composition of two automata is not necessary an FSM language. For this reason, the obtained language should be intersected with the language $(IO)^*$, where I and O are external input and output alphabets of composition, to ensure that each input is followed by some output.

Given FSMs $S = \langle S, I_1 \cup V, U \cup O_1, s_0, \lambda_S \rangle$ and $P = \langle P, I_2 \cup U, V \cup O_2, p_0, \lambda_P \rangle$, the parallel composition $C = S \lozenge P$ is derived in the following way. We first derive corresponding automata $Aut(S)$ and $Aut(P)$ and the parallel composition $Aut(S) \lozenge Aut(P)$. The obtained automaton then is intersected with the automaton that accepts the language $(IO)^*$ and is transformed to the FSM C coupling inputs with the following outputs. FSM $C = \langle C, I, O, c_0, \lambda_C \rangle$ is the parallel composition of S and P, where $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$.

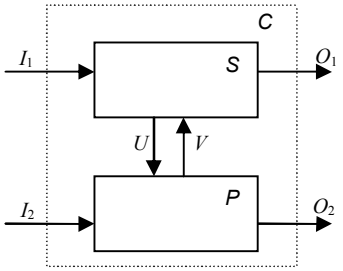

Figure 1 – Parallel composition of FSMs S and P.

It is proven [7], that parallel composition describes following interaction of composed FSMs S and P (Figure 1). The system starts it work, when both S and P are in their initial states, i.e. composition $C = S \lozenge P$ is in its initial state. External environment applies input action either on channel I_1 or I_2, but only one at a time, and then waits for an external output reaction of the system through the one of the output channels O_1 or O_2. The component FSM, which just have got an input action, processes this input and produces either an external output (and so external environment can apply its next input action), or an internal output action that is internal input action for another component FSM. In the latter case, the second component FSM processes a submitted internal input and produces either an external output or an internal output applied to the first component FSM. The dialog between component FSMs continues until one of them produces an external output. When an external output is produced the system is ready to accept the next external input. Here we notice that there can be an external input initiating an infinite dialog between component FSMs. Such infinite cycles of internal actions are called livelocks. However, in practical situations, except of some special cases, input sequences inducing livelocks are usually forbidden.

D. Correspondence between Timed Finite State Machine and Finite State Machine

Before we propose how to construct the parallel composition of timed finite state machines, we introduce the transformation procedure of a TFSM into an FSM and back, and then prove, that obtained FSM correctly describes f-language of the TFSM.

Given TFSM $S = \langle S, I, O, s_0, \lambda_S, \Delta_S \rangle$, we can build an FSM with similar set of functional traces by adding designated input $1 \notin I$ and output $N \notin O$ [8]. Corresponding FSM $A_S = \langle S \cup S_t, I \cup \{1\}, O \cup \{N\}, s_0, \lambda_S^{\Delta} \rangle$ can be built by adding $T - 1$ copies for each state $s \in S$ with defined a finite time-out $T > 1$. There is a chain of transitions between these copies marked with special input-output symbol $1/N$. All other transitions are preserved for each copy. Formally, constructing of A_S can be done by the use of the following rules:

1. S_t contains all such states $\langle s, t \rangle$, $t = 1, ..., T - 1$ where $s \in S$ and $\Delta_S(s) = (s', T)$, $1 < T < \infty$.

2. For each $s \in S$ and $\langle s, t \rangle \in S_t$ and for each i/o, $i \in I$, $o \in O$, there are transitions (s, i, s', o), $(\langle s, t \rangle, i, s', o) \in \lambda_S^{\Delta}$ iff there is a transition $(s, i, s', o) \in \lambda_S$.

3. For each $s \in S$ such that $\Delta_S(s) = (s, \infty)$ there is a transition $(s, 1, s, N) \in \lambda_S^{\Delta}$.

4. For each $s \in S$ such that $\Delta_S(s) = (s', T)$, $T = 1$, there is a transition $(s, 1, s', N) \in \lambda_S^{\Delta}$.

5. For each $s \in S$ such that $\Delta_S(s) = (s', T)$, $1 < T < \infty$, there are transitions $(s, 1, \langle s, 1 \rangle, N) \in \lambda_S^{\Delta}$; for each $j = 1, \ldots, T - 2$ there are transitions $(\langle s, j \rangle, 1, \langle s, j + 1 \rangle, N) \in \lambda_S^{\Delta}$ and $(\langle s, T - 1 \rangle, 1, s', N) \in \lambda_S^{\Delta}$.

We use S_A to denote $S \cup S_t$, I_A to denote $I \cup \{1\}$ and O_A to denote $O \cup \{N\}$.

By construction, when a given TFSM S has n states, a corresponding FSM A_S has $\sum_{s \in S} n(s)$ states, where $n(s) = 1$ for $\Delta_S(s) = (s, \infty)$ and $n(s) = T$ for $\Delta_S(s) = (s', T)$.

Consider an example in Figure 2. State q of TFSM has timeout 2 and therefore, we add one copy of $\langle q, 1 \rangle$ (denoted "$q1$") which is $1/N$-successor of the state q while its $1/N$-successor is s. The sets of successors of q and $\langle q, 1 \rangle$ for all other I/O pairs coincide.

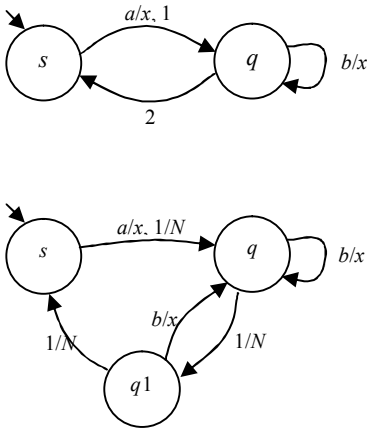

Figure 2 – TFSM S (top figure) and corresponding FSM A_S (bottom figure)

An FSM $A_S = \langle S_A, I_A, O_A, s_0, \lambda_S^{\Delta} \rangle$ such that there are no transitions marked with $1/o$ or i/N where $o \neq N$ and $i \neq 1$ can be transformed to TFSM $S = \langle S, I, O, s_0, \lambda_S, \Delta_S \rangle$ using the following rules:

1. $\Delta_S(s) = (s, \infty)$ iff $(s, 1, s, N) \in \lambda_R^{\Delta}$.

2. Define $\Delta_S(s) = (s_T, T)$ for all such s that there is a chain of transitions $s \xrightarrow{1/N} s_1 \xrightarrow{1/N} \ldots \xrightarrow{1/N} s_{T-1} \xrightarrow{1/N} s_T$, $s, s_T \in S_A$, $T \geq 1$ and for each $i/o \in I \times O$ and $1 \leq j \leq T - 1$ it holds that $suc_S^{\Delta}(s_j, i, o) = suc_S^{\Delta}(s, i, o)$, but for some $i/o \in I \times O$ it holds that $suc_S^{\Delta}(s, i, o) \neq suc_S^{\Delta}(s_T, i, o)$.

3. For each $s \in S_A$, $i \in I$ and $o \in O$, if $(s, i, s', o) \in \lambda_S^{\Delta}$ then $(s, i, s', o) \in \lambda_R$.

Notice that transformation from a given TFSM to an FSM according to the above rules is unique whereas the back transformation from an FSM to a TFSM could be made in different ways; however all such TFSMs are pairwise equivalent, i.e. their f-languages are the same (see the Corollary 2 to Proposition 1).

The following statements establish the relationship between a TFSM and the corresponding FSM built by the above rules.

Proposition 1. FSM A_S has a trace $\underbrace{1/N \ldots 1/N}_{t_1} \; i_1/o_1 \ldots \underbrace{1/N \ldots 1/N}_{t_m} \; i_m/o_m$ iff TFSM S has a functional trace $\langle i_1, t_1 \rangle / o_1 \ldots \langle i_m, t_m \rangle / o_m$.

Proof. According to the rules of constructing A_S, for each two states s_1 and $\langle s_1, j \rangle$ and each $i \in I$ and $o \in O$, the set of i/o-successors of $\langle s_1, j \rangle$ coincides with the set of i/o-successors of state s_1 in S. Thus if there exists such s_2 that $(s_1, i, s_2, o) \in \lambda_S$, then there is a transition $s_1 \xrightarrow{i/o} s_2$ in both machines S and A_S and there is a transition $\langle s_1, j \rangle \xrightarrow{i/o} s_2$ in A_S. Therefore, it is enough to show that A_S is moving from state s_1 to some state $q \in S \cup S_t$ under the sequence $\underbrace{1/N \ldots 1/N}_{t}$ with $t > 0$, and the set of i/o-successors of q in A_S coincides with the set of i/o-successors of the state $time_S(s_1, t)$ in S.

If $\Delta_S(s_1) = (s_1, \infty)$, then $time_S(s_1, t) = s_1$ holds for each value of t; therefore, there is a transition $(s_1, 1, s_1, N) \in \lambda_S^{\Delta}$ and A_S remains at state s_1 under the sequence $\underbrace{1/N \ldots 1/N}_{t}$.

Consider now $\Delta_S(s_1) = (s_T, T)$. If $t < T$, then $time_S(s_1, t) = s_1$ and the sequence $\underbrace{1/N \ldots 1/N}_{t}$ moves A_S from

s_1 to $\langle s_1, t\rangle$. The set of i/o-successors of $\langle s_1, t\rangle$ in A_S coincides with the set of i/o-successors of s_1 in S. If $t \geq T$, then $time_S(s_1, t) = time_S(time_S(s_1, T), t - T) = time_S(s_T, t - T)$. By construction, the sequence $\underbrace{1/N \ldots 1/N}_{T}$ moves A_S from s_1 to s_T, and the sequence $\underbrace{1/N \ldots 1/N}_{t-T}$ is applied to A_S at state s_T, i.e., this case is inductively reduced to the previous case $t < T$.□

Corollary 1. TFSMs S and P are equivalent iff corresponding FSMs A_S and A_P are equivalent.

Corollary 2. If TFSMs S and P both are built by the above procedure from an FSMs A_S, then S and P are equivalent.

Proposition 2. TFSM S is deterministic (complete or observable) iff the corresponding FSM A_S is deterministic (complete or observable).

Proof. The property to be deterministic, observable and complete is specified by the cardinality of sets of i/o- and i-successors. FSM A_S has one and only one transition with pair $1/N$ at each state, that is why properties of FSM A_S to be deterministic, observable and complete depend on transitions with other I/O pairs.

By construction it holds that $suc_S^{\Delta}(\langle s, t\rangle, i, o) = suc_S^{\Delta}(s, i, o) = suc_S(s, i, o)$ for each state s, $\Delta_S(s) = (s', T)$, and for any value of $t < T$. Hence $|suc_S^{\Delta}(\langle s, t\rangle, i, o)| = |suc_S^{\Delta}(s, i, o)| = |suc_S(s, i, o)|$ and $|suc_S^{\Delta}(\langle s, t\rangle, i)| = |suc_S^{\Delta}(s, i)| = |suc_S(s, i)|$. □

E. Parallel composition of TFSMs

Parallel composition of two TFSMs S and P is a TFSM $C = S \lozenge P$ obtained from the FSM $A_S \lozenge A_P$.

Let us illustrate our approach by constructing the parallel composition of TFSMs.

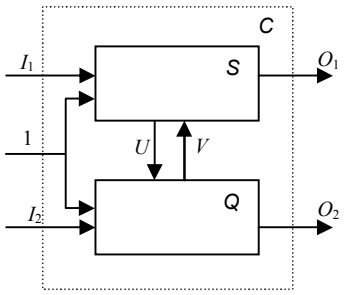

Figure 3 - Parallel composition of TFSMs as a parallel composition of corresponding FSMs

The parallel composition of FSMs that corresponds to the parallel composition of TFSMs is shown in Figure 3. In this case, port 1 is a common port for both machines as it corresponds to a counter of ticks and this accepts the designated input 1 that is an input for both component FSMs and can be considered as an input that synchronizes time behaviour of component FSMs. The designated output N is observed, when there are no outputs at ports O_1 and O_2 (it is observed at both of the ports). Each component FSM has its own time variable, which increments every moment when component gets the designated input 1, and since this signal is applied via a common port for both components the global time is used, and thus, we can say that it synchronizes the behaviour of component FSMs.

As an example, consider the composition of TFSM S in Fig. 2 and P in Fig. 4 where corresponding FSMs are shown as bottom figures. Consider symbols a and o to be external input and output respectively, x and b are internal symbols.

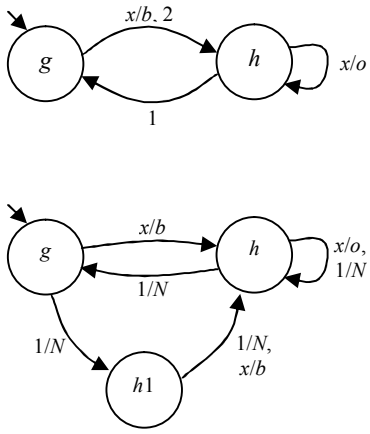

Figure 4 – TFSM P (top figure) and corresponding FSM A_P (bottom figure)

To derive the parallel composition of FSMs, we firstly construct the related automata which are shown in Figure 5. Double lines denote accepting states.

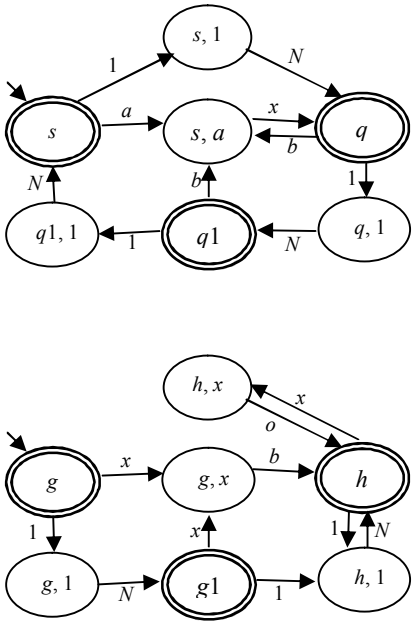

Figure 5 – Automata $Aut(A_S)$ (top figure) and $Aut(A_P)$ (bottom figure)

The second step is to derive the intersection of expended automata that is shown in Figure 6. This intersection should be restricted onto external alphabet $(I \cup \{1\} \cup O \cup \{N\})$ and this restriction intersected with an automaton that accepts the language $[(I \cup \{1\})(O \cup \{N\})]^*$ and it is shown in Figure 7.

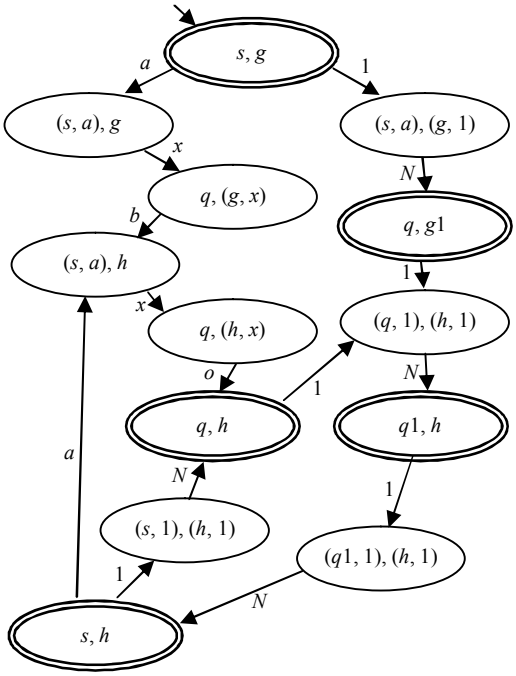

Figure 6 – Intersection of $Aut(A_S)$ and $Aut(A_P)$

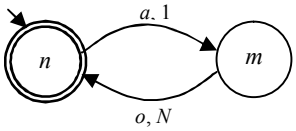

Figure 7 – an automaton accepting language $[\{a, 1\}\{o, N\}]^*$

We then derive a corresponding FSM coupling inputs and the following outputs (Figure 8) and transform this FSM to a corresponding TFSM (Figure 9) that is the parallel composition of TFSMs S and P.

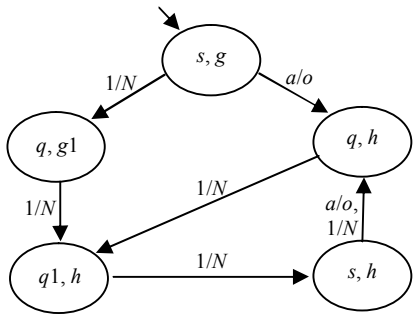

Figure 8 – Composition of S and P (FSM)

The state $(q1, h)$ is copy of the states (q, h) and $(q, g1)$, so there is a time-out equals 2 in the states (q, h) and $(q, g1)$. Furthermore, the states (q, h) and $(q, g1)$ are (f-)equivalent likewise the states (s, g) and (s, h). That is why we keep only two states in TFSM, shown in Figure 9.

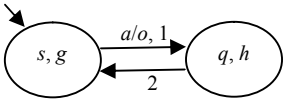

Figure 9 – Composition of S and P (TFSM)

IV. Conclusion and Future Work

The propositions 1 and 2 with corollaries give an approach for solving different problems of TFSMs: first, the corresponding FSMs should be constructed, then appropriate methods of FSM theory can be applied to solve the problem of interest and, finally, the result should be converted back to a TFSM. In this paper we used this approach to define, but more importantly, to construct parallel composition of given TFSMs. However, there is a weak point in the presented work. We have not given a proof of the fact, that such a way to construct parallel composition gives a TFSM which describes a system, combined from two TFSMs, operating in the slow environment setting, as it is done for FSM parallel composition [7]. But Propositions 1 and 2 give confidence, that such a proof can be obtained.

Another direction of research with proposed approach, which we want to designate, is solving the TFSM equations. This line of researches is not covered enough in works on timed finite state machines and we believe that known methods for solving the FSM equations can be adapted to TFSMs easily enough.

References

[1] М. Громов, Н. Евтушенко, “Синтез различающих экспериментов для временных автоматов,” Программирование, № 4, Москва: МАИК, 2010, с. 1–11.

[2] M. Gromov, K. El-Fakih, N. Shabaldina and N. Yevtushenko, “Distinguishing non-deterministic timed finite state machines,” in FMOODS/FORTE-2009, LNCS, vol. 5522, Berlin: Springer, pp. 137–151, 2009.

[3] M. G. Merayo, M. Nunez and I. Rodrigez, “Formal testing from timed finite state machines,” in Computer Networks, vol. 52(2), 2008, pp. 432–460.

[4] K. El-Fakih, N. Yevtushenko and H. Fouchal, “Testing finite state machines with guaranteed fault coverage,” in TESTCOM/FATES-2009, LNCS vol. 5826, Berlin: Springer, pp. 66–80, 2009.

[5] A. V. Aho and J. D. Ulman, “The theory of parsing, translation and compiling: Parsing,” New Jersey:Prentice-Hall, 1002 p., 1973.

[6] A. Gill, “Introduction to the theory of finite-state machines,” New-York:McGraw-Hill, 207 p., 1962.

[7] Спицына Н. В. Синтез тестов для проверки взаимодействия дискретных управляющих систем методами теории автоматов: Диссертация на соискание ученой степени канд. технических наук. – Томск, 2005. – 158 с.

[8] М. Жигулин, Н. Евтушенко, И. Дмитриев, “Синтез тестов с гарантированной полнотой для временных автоматов,” Известия Томского политехнического университета, Т. 316, № 5, Томск: Издательство ТПУ, с. 104–110, 2010.

[9] J. Tretmans, “Test geniration with inputs, outputs and repetitive quiescence,” Software–Concepts and Tools, vol. 17(3), pp. 103–120, 1996