Timed Synchronizing Sequences

N. Giambiasi, C. Frydman
{norbert.giambiasi, Claudia.frydman}@univ-amu.fr

LSIS UMR CNRS 7296
Aix-Marseille University
and
CIFASIS
CONICET-UNR-AMU
Rosario, Argentina

Keywords: state identification, discrete events, sequential machines, synchronizing sequences

Abstract

State-identification experiments are designed to identify the final state of a DES, which is seen as a black box, when its initial state is unknown. A classical solution to this problem constitutes determining a Synchronizing Sequence or a Homing Sequence. In this paper, we show that some classical methods for state identification in untimed sequential machines can be easily extended to Timed Sequential Machines.

1. INTRODUCTION

Extensive theory is available on timed discrete event formalisms such as the Discrete Event Specification: DEVS [1] [2], G-DEVS [3] and Timed Automata [4-7]. These classes of formalisms have an important expressive power that allows complex timed behavior to be specified, verified and simulated [8]. Less expressive timed formalisms can be useful in some classes of applications, such as the design of manufacturing control systems and asynchronous circuit design [9]. In addition, minimization and testing methods [6, 10] can easily be extended to these less expressive formalisms. In this way, we have proposed a new formalism, called Timed Sequential Machines. TSMs [11] are a sub-class of DEVS that have a finite number of states. In addition, the next state of a TSM depends on the elapsed time in the current state.

In this paper, we show that some classical methods for state identification of untimed sequential machines can be easily extended to TSM. State-identification experiments are designed to identify the final state of a DES, which is seen as a black box, when its initial state is unknown [10, 12]. A classical solution to this problem constitutes determining a Synchronizing Sequence (SS) or a Homing Sequence (HS). This problem was essentially solved completely in the 1960's by using untimed formalisms (finite state machines or automata) to model the considered DES.

Extensive theory is available on state identification for classical finite state machines. Therefore, ordinary finite state machines are not sufficiently powerful to model physical systems in an accurate way or to propose practical solutions to some classes of problems, such as state identification. In this paper, we propose a first approach to developing state identification experiments that have a timed formalism: Timed Sequential Machines.

After a recall on Timed Sequential Machines, we propose a method for building a class of timed successor trees of a TSM. We show that synchronizing procedures that are developed on classical sequential machines can be extended to the timed aspects of TSMs [10, 16]. We propose new definitions and methods to construct synchronizing sequences of TSMs, and some examples are given to illustrate these methods.

Some applications fields for this work are those for the test of discrete event control systems and the design and test of asynchronous sequential circuits.

2. TIMED SEQUENTIAL MACHINES (TSM) [11]

2.1. Definition

A Timed Sequential Machine (TSM) is a structure that is defined as follows.

$$M = < X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >,$$

$X = \{x_i\}$ is a finite set of input events.

$Y = \{y_i\}$ is a finite set of output events.

$$S = \left\{ s_i = \left(p_i, \sigma_i \right) \wedge p_i \in V, \sigma_i \in \mathbb{R}_+ \bigcup \{\infty\} \right\}$$

is a finite set of states, where σ_i represents the lifetime of state s_i, p_i is the name of the state or of a subset of states, V is any finite set of symbols or integers, and p_i is called the *phase* of the model. Here, σ_i is a constant that is associated with a state, when the number of states is finite, and the number of σ_i is finite.

A state s_i with an infinite lifetime is said to be a *steady* or *stable* state. A state with a finite lifetime is a *transitory* state. Denoting S_S as the subset of steady states and S_T as the subset of transitory states, we have the following:

$$\sigma_i = \infty \Leftrightarrow s_i \in S_S, \sigma_i \in \mathbb{R}_+ \Leftrightarrow s_i \in S_T,$$

$$S = S_S \bigcup S_T, S_S \bigcap S_T = \varnothing.$$

$\delta_{ext} : S \times X \to S$ is the external transition function that specifies the state changes that are due to the input (in other words, external) events.
$\delta_{int} : S_T \to S$ is the internal transition function that defines the state changes that are due to an internal event that corresponds to autonomous behavior; in other words, an internal event occurs after the lifetime of the current state has elapsed.

$\lambda : S_T \to Y$, λ is the output function that is defined only for transitory states.

The next state of a TMS does not depend on the lifetime of the current state:

$$\forall \sigma_j \in \mathbb{R}_+ \bigcup \{\infty\}, \delta_{ext}((p_i, \sigma_j), x_i) = s_l,$$
$$\forall \sigma_j \in \mathbb{R}_+, \delta_{\text{int}}((p_i, \sigma_j)) = s_l.$$

Similar to the DEVS formalism [1], we introduce the definition of the total state $q_i = (s_i, e)$, where e is the elapsed time in the current state s_i .

$$Q = \{(s_i, e), s_i \in S, e \in \mathbb{R}_+^0 \bigcup \{\infty\}\}.$$

The elapsed time e is reset to zero when a discrete transition occurs.

We also introduce a lifetime function, which yields the lifetime of a state, as follows.

$$lifetime : S \to \mathbb{R}_+ \bigcup \{\infty\}.$$

The *lifetime* function defines the maximum time that the model can remain in the current state.

$$lifetime\,(s_i) = \sigma_i$$

Remark: The output function is defined only for transitory states because we assume that no response of a real system can be instantaneous.

We first provide an informal interpretation of the TSM structure.

- If the model is in state s, then the amount of time that has elapsed since the last event is e, where
$$0 \le e \le lifetime(s).$$

- If no external event occurs, then the model remains in this state s until e = *lifetime(s).* At this time, if *lifetime(s) is finite,* then an output event $\lambda(s)$ is emitted, and the new state is $s' = \delta_{int}(s)$. This state change is called an internal transition or an autonomous transition.

- If an input event x occurs before an internal transition, then the model instantly changes to the state $s' = \delta_{ext}(s,x)$. This state change is called an external transition.

In the case of simultaneous internal and external events, the supervisor of the atomic simulator first sends in the internal event. Subsequently, the external event is treated with the same simulation time, and the next state is defined as follows: $s' = \delta_{ext}(\delta_{\text{int}}(s_i), x_i)$.

A TSM can be represented as a classical sequential machine by a state transition diagram or a transition table that indicates the lifetime of the states. In the state transition diagram of a TSM, the vertices represent the states that have the value of a state lifetime, and the edges are labeled with the input event or the output event that is associated with the transition. In the case of a steady state, the lifetime is omitted.

2.2. Definition: Legitimate TSM

$M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ *defines a legitimate TSM if*

- a lifetime is defined for every state,
- the internal transition function is defined for every transitory state, and
- the lifetime of any state is greater than 0,
$$\forall s_i \in S, lifetime(s_i) = \sigma_i \Rightarrow \sigma_i > 0 \ .$$

2.3. Execution Fragments and Traces of a TSM

We introduce formal notion for the execution of a TSM and its traces to formally specify the full behavior of a TSM. These concepts are analogous to those concepts that are commonly used to describe the behavior of Timed Automata [7, 12].

First, we define the concept of the timed execution fragment of a TSM. A timed execution fragment, which is denoted as $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$, is a finite alternating sequence of state trajectories v_i and discrete events z_i.

2.3.1. Definition: Execution Fragment of a TSM Model

An *execution fragment* of a TSM model $M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ is a finite alternating sequence $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ that satisfies the following criteria.

1. Pure time passage.

Each v_i is a function from a real interval of the form $I_i = [0, t_i]$ to the set Q of total states of M such that

$$\forall j, j' \in I_i \mid j < j' \text{, if } v_i(j) = (s, e) \text{ ,}$$

$$\text{then } v_i(j') = (s, e + j' - j) \ .$$

2. Discrete event transition.

Each z_i is an input or an output event, and if $(s_k, e) = v_{i-1}(sup(I_{i-1}))$, and $(s_l, 0) = v_i(inf(I_i))$, then one of the following conditions holds.

- $z_i \in Y , \delta_{int}(s_k) = s_l,$
$lifetime(s_k) = e \wedge \lambda(s_k) = z_i$, or

- $z_i \in X, \delta_{ext}(s_k, z_i) = s_l$.

Every discrete event transition is associated with a discrete event, which is either

- an output event that corresponds to an internal transition (i.e., a first-case scenario), or

- an input event that corresponds to an external transition (i.e., a second-case scenario).

A timed execution fragment describes all of the discrete state changes that occur and the evolution of the state during time-passage transitions.

2.3.2. Definition: Execution of a TSM

Let $M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ be a TSM that has an initial state s_i, where $s_i \in S$. Then, we call an *execution* of *M* an execution fragment of *M* that begins with s_i.

We use the following definitions:

- $execs^*(M)$ is the set of all finite executions of *M* (a finite number of events),

- $execs(M)$ is the set of all of the executions of *M*.

-*state(α)* is the set of discrete states that appear in the execution fragment *α*.

- *firststate(α)* and *laststate(α)* are the respective functions that indicate the first discrete state and the last discrete state of a finite execution fragment $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$.

$$firststate(\alpha) = s_i \ if \ v_0\left(\inf(I_0)\right) = (s_i, 0)$$

$$laststate(\alpha) = s_i \ if \ v_n\left(\sup(I_n)\right) = (s_i, e).$$

2.3.3. Definition: Trace of a TSM

Let $M = < X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ be a TSM, and let $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ be an execution fragment of *M*. Then, *trace(α)* is defined to be the tuple (θ_I, θ_O, w) such that θ_I and θ_O are sequences that are composed of all pairs of input and output events of α , respectively, and their times of occurrences in α are sorted in chronological order. θ_I and θ_O are the respective input and output traces of α . Here, *w* is the total time of execution, which is defined as follows:

$$w = \sum_{0 \le j \le n} sup\left(I_j\right).$$

Formally, the time of occurrence of an event z_i of $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ is equal to

$\sum_{0 \le j \le i} sup\left(I_j\right)$, where I_j is the domain of v_j .

The set of all of the finite traces of a TSM is defined as follows:

$$traces(M) = \left\{ trace(\alpha), \middle| \alpha \in execs^*(M) \right\}.$$

We denote $length(\theta)$ as the function that provides the number of events in the input (or output) traces θ .

2.3.4. Definition: Trace of a TSM

We introduce the concept of a timed relative input trace to refer to the relative time of occurrence of an event with respect to the previous input event (and not the absolute occurrence time). In other words, given a timed input trace $\theta_I = \langle (x_0, t_0), (x_1, t_1), \ldots, (x_n, t_n) \rangle$, we refer not to θ_I , but to *rel* (θ_I), where the function *rel* is defined as follows:

$$rel(\langle (x_0, t_0), (x_1, t_1), \ldots, (x_n, t_n) \rangle)$$
$$= \langle (x_0, 0), (x_1, t_1 - t_0), \ldots, (x_n, t_n - t_{n-1}) \rangle$$

It is easy to show that *rel* is bijective; as a result, it is equivalent to use either θ_I or $rel(\theta_I)$.

2.3.5. Definition: Fast Execution Fragment *(FEF)*

A timed execution fragment $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ of a *TSM* $M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ is a *fast execution fragment* (FEF) iff

$z_i \in \alpha \Rightarrow z_i \in X$

No internal event appears in a fast timed execution fragment. Notice that

- for a given TSM and for its *FEFs*, all of the transitory states have a behavior that is equivalent to stable states.

- if $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ is a fast execution fragment and $laststate(\alpha) = s_n$, then

$s_n \in S_S \vee (s_n \in S_T \wedge (\sup(I_n) < lifetime(s_n)))$.

In the following, for a TSM *M,* we note that

- $exec_w^{\theta_I^k}(M)$ is the set of finite executions that have the same input trace θ_I^k and a total time of execution that is equal to *w*, and

- $exec_{Fast}{}^*(M)$ is the set of finite and fast executions of *M*.

2.3.6. Theorem

A timed execution fragment $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ of a *TSM* $M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ is a *fast timed execution fragment* if and only if

$\forall v_i \in \alpha, v_i = (s_i, e) \wedge s_i \in S_T \Rightarrow \sup(I_i) < lifetime(s_i)$

proof:

- *Necessary:* no output event can occur in an FTEF, which implies that an input event occurs before the elapsed time in the current transitory state becomes equal to the lifetime of the current state.

- *Sufficient:* if the elapsed time in the current transitory state is lower than its lifetime, then the internal transition does not occur, and no output event is sent out.

2.3.7. Theorem

For a *TSM* $M =< X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$, any execution fragment $\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ with $trace(\alpha) = (\theta_I, \theta_O, w)$ and $rel(\theta_I) = \langle (x_0, \tau_0), (x_1, \tau_1), \ldots, (x_n, \tau_n) \rangle$

such that:

$$\forall (x_i, \tau_i) \in \theta_I, \tau_i < \min(lifetimes(s_i))$$

is a *fast execution fragment*.

proof: The proof is obvious. If the elapsed time between two successive input events is smaller than the minimum of the transitory state lifetimes, then no internal transition can occur, and no output event is sent out.

2.3.8. Definition: Fast Input Trace

An input trace $\theta_I^{\alpha^k}$ of a TSM M is a *fast input trace* for an execution fragment α^k if

$$\exists \alpha^k \in exec^*_{fast}(M) \Big| trace(\alpha^k) = (\theta_I^{\alpha^k}, \theta_O, w).$$

Notice that

$$trace(\alpha) = (\theta_i, \theta_O, w) \wedge \alpha \in exec^*_{fast}(M) \Rightarrow \theta_O = \varnothing.$$

A fast input trace is defined for a given execution fragment, in other words, for a given initial state.

2.3.9. Definition: Total Fast Input Trace

An input trace $\theta_I^{exec^*_{fast}}$ is a *total fast input trace* of a TMS M if it is a fast input trace for every initial state of *M:*

$$\forall s_k \in S, \exists \alpha_i^k \in exec^k_{fast} \text{ with } firstate(\alpha_i^k) = s_k,$$

such that:

$$trace(\alpha^k) = (\theta_I^{exec^*_{fast}}, \theta_O, w) \Rightarrow \theta_O = \varnothing.$$

A total fast input trace is a sequence of events that does not produce any output events regardless of the initial state of the TSM.

3. SYNCHRONIZING INPUT TRACE [12]

A real system modeled by a TSM is in an unknown initial state, and we want to perform a finite execution of it to drive the system into a given final state s_l. The corresponding input trace is called a *synchronizing input trace* for the target state s_l. A *synchronizing input trace* takes a TSM to the same final state, regardless of the initial state and the outputs.

3.1. Definition: Synchronizing Input Trace

Let us consider an execution fragment α_i that has $trace(\alpha_i) = (\theta_I^k, \theta_O^i, w)$; then, the input trace θ_I^k is a synchronizing *input trace* of length n for s_l iff

$$\forall \alpha_i \in exec_w^{\theta_I^K}(M), laststate(\alpha_i) = s_l.$$

The last discrete states of all of the execution fragments with the same input trace and the same time of execution have the same final state s_l.

3.2. Definition: Fast Synchronizing Input Trace

Let us consider an execution fragment:

$$\alpha = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$$

with $trace(\alpha_i) = (\theta_I^k, \theta_O^i, w)$; then, the input trace θ_I^k is a fast synchronizing input trace of length n for s_l iff

$$\forall \alpha_i \in exec_w^{\theta_I^K}(M), laststate(\alpha_i) = s_l$$

and

$$\theta_O^i = \varnothing.$$

3.2. Timed Successor Tree

The *successor tree* of a sequential machine [10, 12] shows the behavior of the machine starting from an Initial State Uncertainty (*ISU)* under all possible input sequences. For every input sequence, the tree contains a path that starts from the root, and every node is annotated with the corresponding current State Uncertainty (SU).

For a TSM, a successor tree must show the behavior of the TSM from an *ISU* under all possible *timed* input traces. Therefore, for a finite given time of execution and a subset of input events, there are infinity input traces (an infinite number of occurrence times of input events in a finite time interval). It is obvious that it is impossible to build a successor tree that explicitly displays all of the finite executions of a TSM.

For a given TSM, a synchronizing sequence can be obtained (if it exists) by constructing an execution tree by ignoring the output events. However, it remains the problem of an infinite tree for a finite execution time. To avoid this problem, we propose, as a first step, to build a specific execution tree while considering only the fastest input traces. In this tree, a path corresponds to an infinite set of the fastest execution fragments. This timed successor tree is finite for a finite time of execution and a finite number of input events.

3.3. Building a Fast Successor Tree (*FST*) of a TSM

For a given TSM *M*, a given initial uncertainty ISU^i and a total execution time *w,* the corresponding *FST* displays the successor uncertainties only for fast execution fragments. To build this *FST*, we compute all of the SU_l^i -*successor* uncertainties for all of the fastest input traces for the finite given time of execution. For a given state uncertainty SU_i, any time interval between the previous input event and the considered input event is lower than the minimum of the lifetimes of the transitory states in SU_i.

The leaves of the *FST* are labeled by the corresponding vector uncertainty and are arranged in successive levels numbered 0, 1, …, j, … The branches are labeled by the input vectors and the time constraints that are needed to obtain fast input traces.

Each finite path of the fast execution tree describes a set of fast input traces with a maximum execution time *w.*

For example, considering the TSM of figure 1 with an $ISU_0^i = (ABCD)$, we have

$$lifetime(A) = 10\ t.u.,$$
$$lifetime(B) = 25\ t.u.$$

Then, the time constraint for the first input events is that the elapsed time in the current state must be lower than 10 *t.u.* For an input trace of length two, we obtain the *FST* of figure 2.

From the IUS_0, we consider that the input event *a* is applied before the elapsed time becomes equal to 10 *t.u.* because

$$s_i \in IUS_0, \min(lifetime(s_i)) = 10\ t.u.$$

Then, we obtain the uncertainty vector (ABC). Applying the input event *b* to (ABCD) with the same time constraint, we obtain the uncertainty vector (ABCD), and from the leaf (ABC), we obtain the leaves (AC) and (ABCD). We can deduce that a fast input trace with the sequence of events (*a, a*) conduces the system into state A or state C.

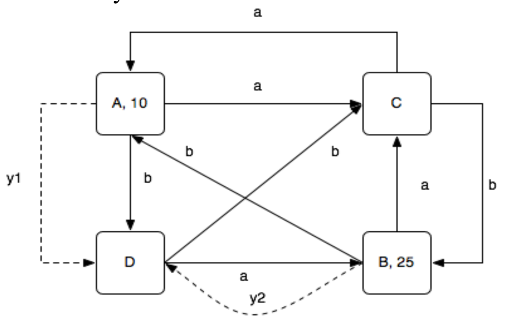

Figure 1: Timed State Transition Diagram of M^1.

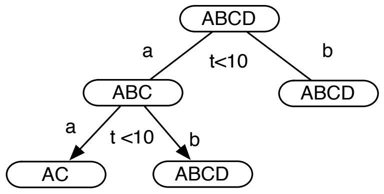

Figure 2: Timed execution tree

3.4. Fast Synchronizing Tree (FST) [10, 12]

Then, the *FST* can be used to define fast *synchronizing* input traces (if at least one exists) by adding two terminal rules to the building process of the *FST*:

- the leaf is associated with an uncertainty that appears in some leaf for a preceding level, and
- the leaf contains an uncertainty with a single element.

A path in this tree displays a set of fastest execution fragments. For a given state uncertainty SU_i, any time interval between two input events is lower than the minimum of the lifetime of the transitory states of SU_i:

3.5. Example

In this example, we built the *FST* (figure3) of the TSM M^1 (figure 1) for a total execution time of $w > 100$ *t.u.* This TSM has the timed state transition diagram given in figure 3.

The initial uncertainty being $ISU_0 = (ABCD)$, the *FST* is built considering that the elapsed time τ in the states of $ISU_0 = (ABCD)$ is such that

$$\tau < \min(lifetime(s_k)), s_k \in IUS_0 \wedge s_k \in S_T,$$

Additionally, in this case, $\tau < 10\,t.u.$

Then, for the input events a and b, we obtain the following successors for IUS_0 :

$$SU_1^a = (ABC)$$
$$SU_1^b = (ABCD)$$

SU_1^b is a terminal node because it has the same label as a previous node.

For $SU_1^a = (ABC)$, the minimum of the lifetime for the transitory states is still 10 *t.u*. The elapsed time before the input event occurrences must be lower than 10 *t.u*., and so on.

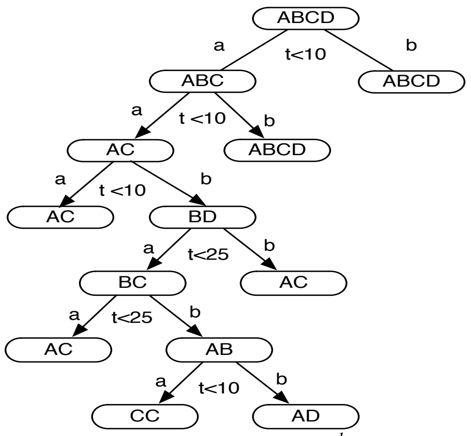

Figure 3: FST of M^1

On this *FST,* we deduce that the path beginning with (ABCD) and ending with (CC) defines a set of fast synchronizing input traces for the target state C.

The input trace,

$$\theta_I =< (a,8)(a,15)(b,24)(a,44)(b,59)(a,64) >,$$

is a fast synchronizing input trace for the target state C.

3.6. Greedy and Cycle Algorithms [13, 14]

The synchronizing tree method has been proposed to provide the shortest "Synchronizing Sequences" (SS). Such a method is suitable for small size systems. In fact, the problem of finding the shortest SSs is known to be NP-complete [13]. Two polynomial algorithms have been used mainly to provide an SS (which is not necessarily the shortest one): the so-called greedy and cycle algorithms. In particular, the greedy algorithm [13] determines an input sequence that takes a given sequential machine, regardless of its initial state, to a known target state: note that the target state is determined by the algorithm and cannot be specified by the user. Greedy and cycle algorithms are based on the construction of an auxiliary graph [16].

For TSMs, a fast auxiliary graph can be defined as follows.

3.6.1. Definition: Fast Auxiliary Graph (FAG)

Let us consider a TSM $M = < X, Y, S, \delta_{ext}, \delta_{\text{int}}, \lambda >$ with n states, and its fast auxiliary graph $FAG(M)$ has $n(n+1)/2$ nodes, one for every unordered pair (s_i, s_j) of states of S, including the pairs (s_i, s_i) of identical states. There is an edge from node (s_i, s_j) to (s_k, s_l) that is labeled with the input event $x_i \in X$ iff

$$\delta_{ext}(s_i, x_i) = s_k \wedge \delta_{ext}(s_j, x_i) = s_l \ .$$

In this definition of the FAG, we do not include the internal transition because we assume that the considered input traces are fast input traces only. Then, the construction is identical to the construction for finite sequential machines.

If, in the auxiliary graph, there is a path from every node (s_i, s_j) to a node (s_i, s_i), then an *FSS* exists for the target state s_i [14].

3.6.2. Example

For the TMS of figure 1, we obtain the auxiliary graph given in figure 4.

From this auxiliary graph, we can see that there is a path from any node to the node (CC); therefore a fast synchronizing input trace exists for the target state C [5].

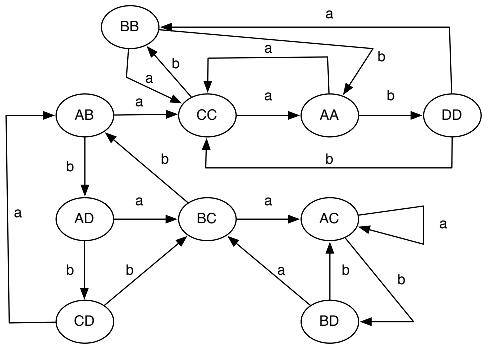

Figure 4: Auxiliary graph of M^1.

3.6.3. Maximal Fast Synchronizing Input Traces

If a fast synchronizing input trace exists for a given TSM M with n states, then its length (number of input events) is at most $n(n-1)^2/2$ [10, 12]. It can be shown by a more careful argument that the length of the constructed synchronizing sequence is at most $n(n^2-1)/6$ [13, 14].

3.6.4. Theorem

Let us consider an execution fragment α with $trace(\alpha) = (\theta_I, \theta_O, w)$ and that θ_I is a fast synchronizing input trace; in that case, if θ_I is a fast synchronizing input trace, then the total time of execution $w_{\max}$ is at most equal to $w_{\max} = N.\max(lifetime(s_i)) + \min(lifetime(s_i)), s_i \in S_T$

with: $N = ((n(n-1)^2/2))$.

Proof: The maximum number of events in the input trace is $(n(n-1)^2/2)$, and the number of time intervals in the execution fragment is $((n(n-1)^2/2)+1)$. Then, the maximum duration of the time intervals between the input events is $\max(lifetime(s_i)), s_i \in S_T$.

The duration after the last event is lower than $\min(lifetime(s_i)), s_i \in S_T$.

Therefore, $w_{\max}$ is at most equal to: $w_{\max} = N.\max(lifetime(s_i)) + \min(lifetime(s_i)), s_i \in S_T$

with: $N = ((n(n-1)^2/2))$.

3.6.5. Example

For the example given in figure 3, the fast synchronizing input trace

$$\theta_I = < (a,8)(a,15)(b,24)(a,44)(b,59)(a,64) >$$

has a length of 6, and the total time of execution should be 70 t.u. (waiting 6 t.u. after the last event).

This TSM has four states. The length of its fast synchronizing input traces must therefore be lower than 18, and the total execution time must be lower than

$$w_{\max} = 9.(25) + 10,$$
$$w_{\max} = 235.$$

$\theta_I = < (a,8)(a,15)(b,24)(a,44)(b,59)(a,64) >$ respects these constraints.

4. CONCLUSIONS

In this paper, the methods for building synchronizing sequences on sequential machines have been extended to Timed Sequential Machines. TSM models represent a useful subset of timed discrete event formalisms, and TSM models have states that can be properly identified and fault checked. The proposed extension for synchronizing sequences is limited to fast executions of TSMs. Future work involves generalizing this first extension to other types of executions and defining preset and adaptive synchronizing experiments on the TSM.

REFERENCES

[1] Zeigler, B.; H. Praehofer; T. G. Kim. 2000. Theory of Modeling and Simulation, 2nd Edition. Academic Press, London.

[2] Zeigler, B. 1984. Theory of Modelling and Simulation. Krieger Publishing Co., Inc., Melbourne, FL, USA.

[3] Giambiasi, N.; B. Escude; S. Ghosh. 2000. "Gdevs: A Generalized Discrete Event Specification for Accurate Modelling of Dynamic Systems." Transaction of S.C.S.I., 17, no. 3: 120–134.

[4] Alur, R.; D.L. Dill. 1994. A Theory of Timed Automata. Theoretical Computer Science, 126: 183–235.

[5] Henzinger, T; X. Nicollin; J. Sifakis; S. Yovine. 1994. "Symbolic Model-Checking for Real-Time Systems." In Information and Computation, vol 111, Issue 2, June 1994, pages 193–244, Else

[6] Lynch, N.A.; F.W. Vaandrager. 1995. "Forward and Backward Simulations – Part II: Timing-Based Systems." Technical Report, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA.

[7] Merritt, M.; F. Modugno; M.R. Tuttle. 1991. "Time–Constrained Automata." In Proceedings on Concurrency Theory (CONCUR '91), volume 527 of LNCS, Jos C. M. Baeten and Jan Frisco Groote, eds., Berlin, Germany, 408–423.

[8] Nance, R.E. 1981. "The Time and State Relationships in Simulation Modeling." Communications of the ACM, 24, no. 4: 173–179.

[9] Dacharry, H.; Giambiasi, N. 2005. "From Timed Automata to Devs Models: Formal Verification." In Proceedings of Spring Sim'05. SCS - The Society for Modeling and Simulation International, 2005.

[10] Kohavi, Z. 1980. Switching and Finite Automata Theory. Computer Science Series. McGraw-Hill Higher Education.

[11] Giambiasi, N. 2009. "TSM: Temporal Sequential Machines." In Proceedings of the 21st European Modeling and Simulation Symposium, SCS, Canary Island, Tenerife.

[12] Giambiasi, N.; D. Llarul; M. Cristea. 2010. "System State Identification using DEVS." In Discrete-event Modeling and Simulation, Wainer, G.A., Mosterman, P.J., eds., CRC Press.

[13] Eppstein, D. 1990. "Reset Sequences for Monotonic Automata." SIAM Journal on Computing, 19: 500–510.

[14] Trahtman, A.N. 2004. "Some results of implemented algorithms of synchronization." In Proceedings of the 10th Journees Mentoises d'inform, Liege, Belgium, September 8-11.

[15] Lee, D.; Yannakakis, M. 1996. "Principles and Methods of Testing Finite State Machines - A Survey." Proceedings of the IEEE, 84, no. 8.