System State Identification using DEVS

N. Giambiasi*, Diego Llarrull*, Maximiliano Cristiá**

** LSIS - Université Paul Cézanne - Marseille, **CIFASIS, CONICET – Rosario.*

Abstract: *In this paper, we propose an approach for system state identification using the DEVS formalism. In other words, extensions to timed models of classical methods developed in the field of sequential machines are defined. The aim of these methods is to deduce information on the states of a system by observing its input/output behavior. In fact, in this paper, we are concerned with the following testing problems: determine the final state after the test, identify the initial state, verify the unknown initial state.*

A possible field of application for this work is the testing of discrete event control systems for which timed considerations are generally needed.

In this first approach, we propose extension of testing methods for a subset of DEVS models, in this subset, the next states of a model does not depend on the elapsed time in the current state. Finally, we briefly show some considerations about the implementation of these testing methods.

1 Introduction

In the 60s, research was done on the problem of testing finite state machines to ensure their correct functioning and to discover aspects of their behavior. Due to its applications in testing communications protocols, the fault detection problem is still studied in the field of finite state machines, and of some of their extensions.

An important class of testing problems is state identification (pioneered in the seminal 1956

paper of Moore)[8]. An extensive theory is available on this topic for finite state machines (see [7] for a survey and Kohavi's book for a good exposition of the major results published in papers on testing problems, in the 60s[4]). The testing problem was and is still studied in the context of timed Automata[1] which are used to model real-time systems with timed considerations[9].

Therefore, ordinary finite state machines are not powerful enough to model physical systems in an accurate way and, even though timed automata are well adapted for high level specifications, the DEVS formalism proposed by B. Zeigler (which can be seen as a general timed extension of finite state machines) seems more suitable for representing accurate timed behavior of dynamics systems. DEVS allows building discrete timed event abstractions of dynamic systems with a clean simulation semantics and a clean correspondence between the real system and the basic concepts of the formalism (such as states, transitory states, state variables, events, etc...).

In this paper, we propose an approach for system state identification using the DEVS formalism. In other words, we would like to deduce information on the states of the system by observing its input/output behavior. In fact, we are concerned with the following testing problems[7]:

- determine the final state after the test,
- identify the initial state,
- verify the unknown initial state.

A possible field of application for this work is the testing of discrete event control systems for which timed considerations are generally needed, and a discrete event abstraction of the system

to be controlled is also needed in order to realize accurate timed analysis by simulation of a coupled model, which is composed of the control system and the system to be controlled.

In (Dacharry and Giambiasi 2005), a formal methodology for the design and verification of discrete event control systems was proposed. Within this methodology, a high-level specification of a control system is given by a network of timed automata, and the corresponding implementation is expressed by a coupled DEVS model[11]. This allows the formal verification of the conformance of components (atomic DEVS models against timed automata) and the conformance of the whole model. Nevertheless, due to the state explosion problem that appears in the verification of models that deal with a dense time base, the automatic verification of the conformance between the high-level and the low-level models cannot be carried out in a large number of real cases. Therefore, a partial automatic validation of the conformance relation between an implementation and its specification can be possible by generating test cases on a high-level specification and applying these tests to the low-level model description or to an implementation. This kind of test, called conformance testing, is not developed in this paper but the methods presented for minimization or for building some types of input sequences constitute a required step for conformance testing.

The paper is organized as follows:

In Section 2 we recall the existing theory, together with the tools and concepts that will be necessary to extend it. In Section 3, we introduce a subset of the DEVS formalism that we take under consideration, and we adapt and extend the existing methods, concepts and definitions to this subset. In Section 4, we propose an extension of the first subset of models in order to enlarge

the spectrum of models to which the theory of fault detection can be applied, and we briefly show some considerations about the implementation of these testing methods. Finally, we conclude the paper.

2 The DEVS formalism

The DEVS formalism allows atomic models (behavioural models) to be specified and coupled together to build more complex models (coupled models or structural models). These coupled models can themselves be used as components of larger coupled models [11], allowing hierarchical descriptions by means of a model-library.

2.1 Atomic DEVS Models

According to the literature on DEVS [12], the specification of a discrete event model is a structure:

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

where X is the set of the external input events, S the set of the sequential (or discrete) states, Y the set of the output events, δ_{int} is the internal transition function that defines the state changes caused by internal events, δ_{ext} is the external transition function that specifies the state changes due to external events, λ is the output function, and the function $ta : S \to \Re_0^+ \cup \{\infty\}$ represents the maximum duration or lifetime of a state, with $\Re_0^+$ representing positive real numbers. Thus, for a given state s_i, $ta(s_i)$ represents the time interval during which the model will remain in the state s_i provided that no external event occurs.

A state is passive when its lifetime is infinite *(*$ta(s_i) = \infty$) and active when the lifetime is a finite real positive number. Denoting S_a the subset of active states and S_p the subset of passive states, we have $S_a \cap S_p = \varnothing$.

[12] introduces the concept of total states *(s,e)* of a model as:

$$TS = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$$

where *e* represents the elapsed time in state s. The concept of total state is fundamental because it permits one to specify a future state based on the elapsed time in the present state.

A key contribution of DEVS lies in decomposing the traditional transition function into two sub-functions—the internal transition function and the external transition function. The internal transition function, defined as $\delta_{int} : S \rightarrow S$, where S refers to the active (transitory) states of the system, permits one to capture the autonomous evolution of the model. When a model has entered in state s at time t_i, it will transition to state $s' = \delta_{int}(s)$ at time $t_i + ta(s)$ provided that no external event occurs during this period. The external transition function, defined as δ_{ext}: $TS \times X \rightarrow S$, reflects the evolution of the model corresponding to externally induced input events. When an external event x occurs and the model is in state s since the "elapsed time" e, the model will transition to state s': $s' = \delta_{ext}(s, e, x)$. The elapsed time, e, is set to zero for each state transition. The output function, λ: $S \rightarrow Y$, defined only for active states, is executed when the state's lifetime is reached. From the simulation perspective, this needs that the output function is executed prior to the internal transition function.

The pseudo-code of an abstract simulator for atomic models is detailed in Figure 1. This abstract simulator runs with a coordinator, which manages a scheduler and sends to the simulator the input events at the event time t_n. In this way the *operational behaviour or semantics* of a DEVS

model is given by its simulator.

2.2 DEVS-atomic-simulator

The DEVS atomic simulator receives, in a chronological order, input events and internal events from a coordinator. The internal events, are created by the DEVS atomic simulator when the model enters in an active state, the time occurrence of the internal event is tn= tl + ta(s), where tl is the present time (time of the transition to the considered active state).

```
when internal event, at time t:
y = λ(s)
send output event (y, t)
s = δint(s)
tl = t
tn = tl + ta(s)
e=0

when receive input event x, at time t:

e = t – tl
s = δext(s, e, x)
tl = t
tn = tl + ta(s)
e=0

end DEVS-atomic-simulator
```

Figure 1: Pseudo-code of a simulator for DEVS atomic models.

2.3 Execution fragments and traces of DEVS Models

We introduce the formal notion of executions or simulation runs of DEVS models, and their traces as an alternative way to formally specify the full behaviour of DEVS models [2]. These

concepts are analogous to the ones that are commonly used for describing the behaviour of Timed Automata.

Definition 2.1: Execution fragment of a DEVS model

*An **execution fragment** of a DEVS model $\mathcal{D}$ is a finite alternating sequence* $\Upsilon = v_0 x_1 v_1 x_2 v_2 \ldots x_n v_n$, *where:*

1. *pure time passage*

   Each v_i is a function from a real interval $I_i = [0, t_i]$ to the set of total states of $\mathcal{D}$, such that $\forall j, j' \in I_i \mid j < j'$, if $v_i(j) = (s, e)$ then $v_i(j') = (s, e + j' - j)$

2. *discrete event transition*

   Each x_i is an input or output event, and if $(s, e) = v_{i-1}(sup(I_{i-1}), (s', 0) = v_i(inf(I_i))$, one of the following conditions hold:

   a. $x_i \in Y_{\mathcal{D}}, \delta_{int\mathcal{D}}(s) = s'$, $ta(s)=e$, and $\lambda(s) = x_i$

   b. $x_i \in X_{\mathcal{D}}, \delta_{ext\mathcal{D}}(s, e, x_i) = s'$, and $e < ta(s)$.

Definition 2.2: Execution of a DEVS model

Let $\mathcal{D} = (X_{\mathcal{M}}, Y_{\mathcal{M}}, S_{\mathcal{M}}, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}}, s_0)$ be an atomic DEVS with the initial state s_0. Then, an execution for $\mathcal{D}$ is a execution fragment of $\mathcal{D}$ that begins with the initial state s_0.

We denote with $execs^*(\mathcal{D})$, $execs^\omega(\mathcal{D})$ and $execs(\mathcal{D})$ the sets of finite, infinite, and all executions of $\mathcal{D}$, respectively.

We denote *last* and *first* the functions giving the states of the first and the last state trajectories of an execution fragment, respectively:

$$last : execs^{*}(\mathcal{D}) \rightarrow S_p$$
$$last(\alpha) = s_j \quad if \; v_n(\mathbf{sup}(I_n)) = (s_j, t_n)$$

and,

$$first : execs^{*}(\mathcal{D}) \rightarrow S_p$$
$$last(\alpha) = s_j \quad if \; v_0(0) = (s_j, 0)$$

Definition 2.3: Trace of a DEVS model

A trace of an execution fragment $\Upsilon = v_0 z_1 v_1 z_2 v_2 \ldots z_n v_n$ *, noted trace* (Υ)*, of a DEVS model* $\mathcal{D}$ *is defined to be a tuple* (θ_I, θ_O, t) *such that* θ_I *and* θ_O *are sequences consisting of pairs of the form* (z_i, *sup*(I_{i-1})) *where* z_i *is an input or and output event, of* (Υ)*, respectively, and their time of occurrence in* (Υ) *sorted in chronological order of occurrence. t is the total time of execution, defined as* $\sum_{0 \le j \le n} (sup(I_j))$*. Formally, the time of occurrence of an event* x_i *of* Υ *is equal to* $\sum_{0 \le j < i} (sup(I_j))$*, where* I_j *is the domain of* v_j*.*

The set of all traces of a DEVS model, noted traces$(\mathcal{D})$, is defined as $\{ \text{trace}(\Upsilon) \mid \Upsilon \in \text{execs}(\mathcal{D}) \}$.

Remark : Note that for every DEVS model $(\mathcal{D})$, and given a sequence of pairs of input events and their respective time of occurrence, θ_I, its total time of execution t, and an initial state, s_0, there exists a unique execution fragment $\gamma = v_0 x_1 v_1 \ldots x_n v_n$, such that $trace(\gamma) = (\theta_I, \theta_O, t)$ and $v_0(0) = (s_0, 0)$.

This concept is known in the DEVS literature as determinism, and as such, all completely specified DEVS models are deterministic in this sense, considering that when an input event (an

external transition) occurs at same time that an internal transition is scheduled, we give the priority to the external transition.

For our test purpose of timed models, we need to introduce the concepts of slow timed execution fragment and slow timed input trace.

Definition 2.4: Slow timed execution fragment

A timed execution fragment $\Upsilon = v_0\, z_1\, v_1\, z_2\, v_2 \ldots z_n\, v_n$ *of a DEVS atomic model* $\mathcal{D} = \langle X, Y, S, P, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$ *is a slow timed execution fragment if:*

$$\forall v_j \in \alpha \text{ with } v_j = (s_k, e) \wedge s_k \in S_a, \sup(I_j) = \mathrm{ta}(s_k).$$

We denote as $execs^*_{slow}(\mathcal{D})$ *as the set of the finite slow timed execution fragments of* $\mathcal{D}$ *.*

Definition 2.5: Slow timed input trace

An input trace θ_{Ii} *is a slow timed input trace iff :*

$$\exists \alpha_i, \alpha_i \in execs^*_{slow}(\mathcal{D}) \wedge trace(\alpha_i) = (\theta_{Ii}, \theta_{Oi}, t)\ .$$

Let us notice that, since only the output events are observed when black-box testing a DEVS model, then in order to be sure to have a slow input trace, we must wait τ time units between two input events with:

$$\tau > \boldsymbol{max}\left(\boldsymbol{ta}(s_j) \mid s_j \in S_a\right).$$

2.4 DEVS based Hierarchy formalisms

In [3], we introduce a hierarchy of discrete event formalisms with increasing complexity, maintaining at the same time the clarity of concepts and cohesion in the structure. The proposed approach maintains cohesion and clarity by using a similar syntax in all proposed formalisms, and by formally specifying its semantic or operational behaviour in order to avoid an incorrect interpretation of the formalism.

The syntax of all the proposed formalisms is based on the Discrete Event System Specification (DEVS) [12]. We have chosen the DEVS formalism since it incorporates a solid mathematical basis along with a well developed simulation infrastructure. Furthermore, it is considered to be a universal formalism for the modelling and the simulation of discrete event systems.

The different formalisms we have proposed range from a formalism based in the widely known concept of sequential machines (automata), to the more expressive DEVS formalism.

For our current purpose we recall, in the following, the two first formalisms the two first formalisms of the hierarchy on which we have adapted testing methods initially proposed for Mealy machines.

2.5 Formalism hierarchy

The first and less expressive formalism we consider is an untimed discrete event formalism (only the occurrence order of events is taken into account).

Definition 2.6: Untimed.DEVS (U.DEVS)

An atomic U.DEVS model is defined as a structure $\mathcal{D} = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda)$ where:

- X: finite set of input events,
- Y: finite set of output events,
- S: finite set of states,
- δ_{int}: internal transition function $(\delta_{\text{int}} : S \to S)$,
- δ_{ext}: external transition function $(\delta_{ext} : S \times X \to S)$,
- λ : output function $(\lambda : S \to Y)$.

In an U.DEVS model, as in classical DEVS, the set of states is partitioned in two sets, S_a and S_p called active and passive state sets respectively, $S = S_a \cup S_p$.

Only external transitions can occur in a passive state of an U.DEVS model (as in classical DEVS), and only internal transitions can occur in an active state. In other words, δ_{ext} is not defined for the active states of an U.DEVS model .

Note that for U.DEVS, we consider that *X, Y* and *S* are finite sets.

Interpretation: U.DEVS

The model remains in a passive state s_i until it receives an input event x_j, the next state is given by $\delta_{ext}(s_i, x_j)$. When the model reaches an active state s_i it instantaneously transits into the next state defined by $\delta_{int}(s_i)$ and emits the output event $\lambda(s_i)$. It is assumed that no input event can

occur in an active state.

```
U.DEVS-atomic-simulator

when internal event, at time t:
if s ∉ Sa then ERROR else
y = λ(s)
send output event (y, t)
s = δ_int(s)

when receive input event x:
if s ∉ Sp then ERROR else
s = δ_ext(s, x)

end U. DEVS-atomic-simulator
```

Figure 2: Pseudo-code of a simulator for U.DEVS atomic models

Remark: it should be clear that U.DEVS are very closely related to classical sequential machines. Therefore, in U.DEVS, the concepts of active (*transitory*) *and passive* (*steady*) *states are clearly identified, which allows 'clean' specifications of models with transitory and steady states.*

The next formalism in the hierarchy [3], is called Simple.DEVS. This formalism introduces the notions of:

- Phase,
- Phase-lifetime.

Phase is a state variable that ranges over a finite set. In fact, to each value of Phase corresponds a subset of the states of the model. Phase takes a finite number of values, and thus the partition of *S* defined by Phase is finite. A lifetime value is associated to each value p_i of Phase. This value is

given by the lifetime function $ta(p_i)$, which can be any positive real number, 0 or infinite. Phases with a 0 or real number lifetime value are called active phases, while the ones with an infinite lifetime are passive phases. Phase-lifetime is another state variable, Phase-lifetime = $ta(p_i)$ which equals the life time of the present value of Phase.

Notice that these definitions are identical to those of classical DEVS, but for S.DEVS, the lifetime function depends only on the state variable Phase and not on the other state variables. In fact, all the states with the same value of Phase belong to the same class and this means that all the states of a class have the same lifetime.

S.DEVS is a restricted formalism in the sense that it is not possible to model the temporal behaviour of any discrete event system. Its main restrictions are:

- The lifetime of a state depends only on one particular state variable, the Phase state variable.
- The output function depends only on Phase.

Definition 2.7: Simple.DEVS (S.DEVS)

An atomic Simple.DEVS model is defined as the structure:

Shouldn't P be a part of the following structure (in yellow) ?

$\mathcal{D}_s = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$ where:

- X: finite set of input events,
- Y: finite set of output events,
- S: set of states, $S = \{s_i = (s_{v1}, \ldots, s_{vn}, Phase, ta) \mid s_{vi} \in V_i \wedge Phase \in P \wedge ta : \Re_0^+ \cup \{\infty\}\}$.
- δ_{int}: internal transitions $(\delta_{int} : S \to S)$,
- δ_{ext}: external transitions $(\delta_{ext} : S \times X \to S)$,
- λ: output function $(\lambda : S \to Y)$,
- ta: lifetime function $(ta : P \to \Re_0^+ \cup \{\infty\})$, $ta(p_i)$ is the lifetime associated to the value p_i of *Phase*.

A Simple.DEVS model is completely specified, if:

- ta is defined for all the possible values of *Phase*,
- δ_{int} and λ are partial functions such that, $\forall p_i \in P$ with $ta(p_i) \neq inf$, $\delta_{int}(s)$ and $\lambda(s)$ are defined,
- δ_{ext} should be a function defined for all states and all input events.

Interpretation: Simple.DEVS

When the model is in the state s_k with the value p_k of *Phase*, the maximum time it remains in the state s_k is given by $ta(p_k)$. If no external event occurs, when $ta(p_k)$ time is elapsed, the model generates the output event $\lambda(s_k)$ and it changes to the state $s_l = \delta_{int}(s_k)$. If before this internal transition an external event x occurs, the system transitions to the state $s_j = \delta_{ext}(s_k, x)$.

Notice that in S.DEVS, the next state does not depend on the elapsed time in the current state.

In the following, for the sake of simplicity we will concentrate only on these two subsets of atomic DEVS models. In addition, and without loss of generality, we consider models for which every state is totally defined by the two state variables, that is, *Phase* and *ta*.

3 Extending Mealy Machines Fault Detection Techniques To U.DEVS

In order to make a progressive approach to the problem, we first present a procedure for obtaining an U.DEVS with the same behaviour than a Mealy machine. The final objective remains to apply to DEVS models some fault detection techniques developed on Mealy Machines.

We recall that a Mealy Machine[4] is formally stated as a quintuple $M = (I, O, S, \delta, \lambda)$ where *I*, *O* and *S* are finite and nonempty sets of input symbols, output symbols, and states respectively, $\delta : S \times I \to S$ is the state transition function and $\lambda : S \times I \to O$ is the output function.

The first step in defining an U. DEVS model with the same behaviour than a given Mealy machine is to add an active state for every transition of the Mealy machine. Then, every state transition of a Mealy machine $\mathcal{M} = (I_{\mathcal{M}}, O_{\mathcal{M}}, S_{\mathcal{M}}, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}})$ that has the form:

$$s_i \overset{x/y}{\to} s_j,\ x \in I_{\mathcal{M}},\ y \in O_{\mathcal{M}},\ s_i, s_j \in S_{\mathcal{M}}$$

is translated into two transitions in the corresponding U.DEVS model $\mathcal{D} = < X_{\mathcal{D}}, Y_{\mathcal{D}}, S_{\mathcal{D}}, \delta_{int_{\mathcal{D}}}, \delta_{ext_{\mathcal{D}}}, \lambda_{\mathcal{D}}, ta_{\mathcal{D}} >$:

- An external transition of the form $s_i \overset{x/-}{\to} s_{i,x}$ becomes $\delta_{extD}(s_i, x) = s_{i,x},\ s_i \in S_{s_{\mathcal{D}}},\ x \in X_{\mathcal{D}}$ and $s_{i,x} \in S_{a_{\mathcal{D}}}$.

- An internal transition of the form $s_{i,x} \xrightarrow{-/y} s_j$ becomes $\delta_{int_{\mathcal{D}}}(s_{i,k}) = s_j$, $\lambda(s_{i,k}) = y$, $s_{i,k} \in S_{a_{\mathcal{D}}}$, $y \in Y_{\mathcal{D}}$ and $s_j \in S_{s_{\mathcal{D}}}$.

An U.DEVS $\mathcal{D} =< X_{\mathcal{D}}, Y_{\mathcal{D}}, S_{\mathcal{D}}, \delta_{int_{\mathcal{D}}}, \delta_{ext_{\mathcal{D}}}, \lambda_{\mathcal{D}}, ta_{\mathcal{D}} >$ and a Mealy machine $\mathcal{M} = (I_{\mathcal{M}}, O_{\mathcal{M}}, S_{\mathcal{M}}, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}})$ have the same behaviour if:

1. both models have the same input and output event sets, $I_{\mathcal{M}} = X_{\mathcal{D}} \wedge O_{\mathcal{M}} = Y_{\mathcal{D}}$,
2. the set of passive states of $\mathcal{D}$ is equal to the set of all states of $\mathcal{M}$: $S_{\mathcal{M}} = S_{p_{\mathcal{D}}}$
3. for each possible transition $\delta_{\mathcal{M}}(s_i, x) = s_j$ of $\mathcal{M}$ there exist two transitions in $\mathcal{D}$:

$$\delta_{extD}(s_i, x) = s_{i,x} \text{ and } \delta_{int_{\mathcal{D}}}(s_{i,x}) = s_j .$$

4. for each possible output event $y \in I_{\mathcal{M}}$ such that $\lambda_{\mathcal{M}}(s_i, x) = y$ it is required an analogous output event in $\mathcal{D}$

$$\lambda_M(s_i, x) = y \Leftrightarrow \lambda_D(s_{i,x}) = y .$$

Definition 3.1: An Untimed.DEVS is a bipartite DEVS iff:

- $\forall s_i \in S_p \wedge \forall x_i \in X, \delta_{ext}(s_i, x_i) = s_k \Rightarrow s_k \in S_a$.
- $\forall s_k \in S_a, \delta_{int}(s_k) = s_i \Rightarrow s_i \in S_p$.

For Untimed Bipartite DEVS (U.B.DEVS), we introduce the concept of a passive transition function, noted δ_{pass}.

Definition 3.2: The passive transition function of an U.B.DEVS is defined as follows: $\forall x_i \in X, s_i \in S_p$**,**

$$\delta_{pass}(s_i, x_i) = \delta_{int}(\delta_{ext}(s_i, x_i)) = s_j \in S_p .$$

Analogously, we also introduce the concept of a passive output function, noted λ_{pass}.

Definition 3.3: the passive output function of an U.B.DEVS is defined as follows:

$\forall x_i \in X, s_i \in S_p$**,**

$$\lambda_{pass}(s_i, x_i) = \lambda(\delta_{ext}(s_i, x_i)) = y_i \in Y$$

3.1 Minimality - State Equivalence

The definition and properties presented in [4] for Mealy Machines can be easily extended to U.B.DEVS, by considering the following theorem:

Theorem 3.1: To every U.B.DEVS model corresponds a Mealy Machine with the same behaviour and vice-versa.

Proof: it is straightforward to prove that, by considering only the passive transition and output functions of an U.B.DEVS, the existence of active states in this model is concealed. Thus, the model externally appears to have the same set of states than its corresponding Mealy machine. Then, each pair of values $[\delta_{pass}(s_i, x_i), \lambda_{pass}(s_i, x_i)]$ corresponds to the pair $[\delta(s_i, x_i), \lambda(s_i, x_i)]$ of a Mealy Machine, which is equivalent to say that they have the same behaviour.

In Figure 3, we give an example of a state graph of a Mealy Machine (Figure 3a) and of the state graph of its corresponding U.B.DEVS (Figure 3b).

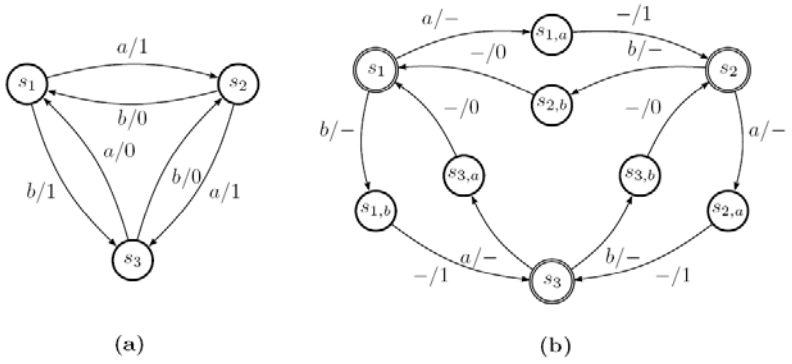


Figure 3: Mealy machine and its corresponding U.B.DEVS.

In order to apply testing methods from Mealy Machines to U.B.DEVS, we need first to define the basic concepts of distinguishable states, state equivalence and state minimization for U.B.DEVS.

Definition 3.4: Distinguishable passive states

Two passive states s_i *and* s_j *of an U.B.DEVS model* $\mathcal{D}$ *are* ***distinguishable*** *if and only if there exist at least two execution fragments* α *and* β *of* $\mathcal{D}$ *with* $\textbf{traces}(\alpha) = (\theta_{I_\alpha}, \theta_{O_\alpha})$, $\textbf{traces}(\beta) = (\theta_{I_\beta}, \theta_{O_\beta})$ *where:*

$$first(\alpha) = s_i,\ first(\beta) = s_j$$

$$\theta_{I_\alpha} = \theta_{I_\beta} \text{ and } \theta_{O_\alpha} \neq \theta_{O_\beta}.$$

Then, θ_{I_α} $(= \theta_{I_\beta})$ *is called a* ***distinguishing sequence*** *of the pair* (s_i, s_j).

If the shortest distinguishing sequence of the pair (s_i, s_j) has a length k, then, s_i and s_j are said to be ***k-distinguishable.***

Definition 3.5: k-equivalence of states

Two ***passive*** *states s_i and s_j of an U.B.DEVS model $\mathcal{D}$ are k-equivalent if and only if they are not k-distinguishable.*

Definition 3.6: State equivalence

Two ***passive*** *states s_i and s_j of an U.B.DEVS model $\mathcal{D}$ are equivalent if and only if they are k-equivalent $\forall k \in \mathbb{N}$.*

Definition 3.7: Equivalence of U.B.DEVS

Two U.M.DEVS models $\mathcal{D}_1$ and $\mathcal{D}_2$ are equivalent if and only if, for every state in $\mathcal{D}_1$, there is a corresponding equivalent state in $\mathcal{D}_2$, and vice-versa.

Definition 3.8: Minimal U.B.DEVS

An U.B.DEVS model is minimal (*reduced*) *if and only if no two states in it are equivalent.*

The concept of minimality is important in the field of fault detection techniques due to the fact that most results are obtained under the hypothesis of minimality. We recall that the minimization procedure is applied under the hypothesis that the machine under consideration is *completely specified*, that is to say, for every state, there is a transition specified for every input

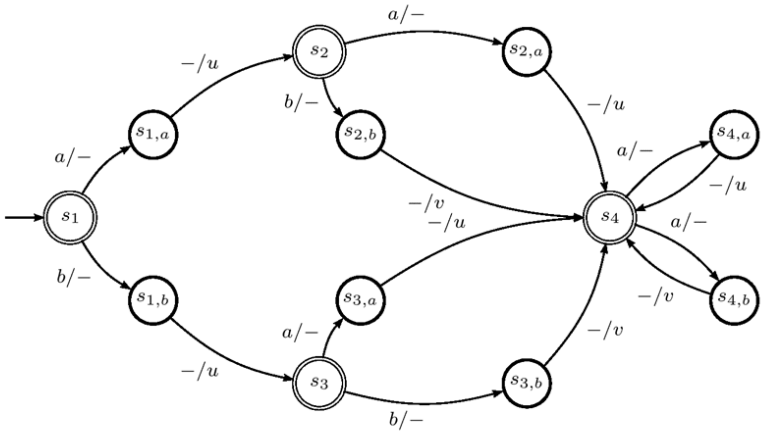
event.

The minimization procedure detailed in [4] can be applied to *completely specified* U.B.DEVS models if only the passive states, the passive transition function δ_{pass} and the passive output function λ_{pass} are considered. Whenever two equivalent passive states are found, it is necessary to remove not only one of these states, but also *every active state to which such state can transition*. Then, every transition that reaches a state that has been deleted has to be redirected to its equivalent state.

Extending the result from Mealy machines we can say that every U.B.DEVS model $\mathcal{D}$ corresponds a minimal U.B.DEVS $\mathcal{D}^*$ which is equivalent to $\mathcal{D}$ and is unique up to isomorphism, provided $\mathcal{D}$ is *completely specified* (the external transition function is defined for all possible input events).

The existence of a minimal form for every completely specified U.B.DEVS model ensures that fault detection techniques defined for Mealy machines can be applied to U.B.DEVS models, after finding its minimal or reduced form.

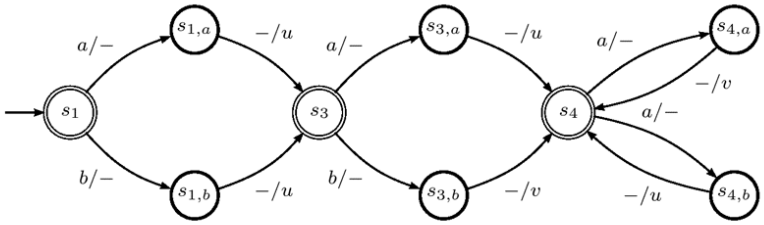
Example: Let us consider the state graph of an U.B.DEVS (Figure 4a), its passive transition table (table with the passive transition function and the passive output function) is represented in (Figure 4b):



(a)

State	$x = a$	$x = b$
s_1	s_2, u	s_3, u
s_2	s_4, u	s_4, v
s_3	s_4, u	s_4, v
s_4	s_4, u	s_4, v

(b)



(c)

State	$x = a$	$x = b$
s_1	s_3, u	s_3, u
s_3	s_4, u	s_4, v
s_4	s_4, v	s_4, u

(d)

Figure 4: An U.B.DEVS model (a) with its associated transition table (b). Resulting minimal U.B.DEVS model (c) after the deletion of s_2 (which is equivalent to s_3), $s_{2,a}$ and $s_{2,b}$, and the resulting transition table (d).

By applying the minimization procedure given in [4], we obtain the minimal U.B.DEVS model by replacing each equivalent class of states by one state. In this example, the states s_2 and s_3 are equivalent, replacing these two states by s_3, we obtain, for this U.B.DEVS model, the state graph and the transition table of Figure 4c and Figure 4d.

3.2 Experiments on U.B.DEVS models

In a testing problem, we consider an implementation (seen as a black-box) of an U.B.DEVS about which we would like to deduce some information by observing its input/output behaviour. An input event sequence is applied to the system and the output sequence is observed in order to infer the needed information. The input sequence can be preset if it is fixed ahead of time, or can be adaptive if the next input event depends on the previously observed outputs.

We will proceed by adapting several concepts such as first distinguishing, homing and synchronizing sequences the basic definitions of distinguishing, homing and synchronizing sequences, in the following, we propose the definitions of experiments adapted to U.B.DEVS from those of Mealy Machines[4].

In the following, we denote with $execs_{Ik}(\mathcal{D})$ *the set of finite executions of* $\mathcal{D}$ *with the same input trace* θ_I^k**.**

Definition 3.9: Distinguishing sequence

Let us consider two execution fragments α *and* β *of an U.B.DEVS* $\mathcal{D}$ *with* $\textbf{traces}(\alpha) = (\theta_{I_\alpha}, \theta_{O_\alpha})$, $\textbf{traces}(\beta) = (\theta_{I_\beta}, \theta_{O_\beta})$ *such that:*

$$first\ (\alpha) = s_i,\ first\ (\beta) = s_j$$

$$and\ \theta_{I_\alpha} = \theta_{I_\beta}$$

Then, θ_{I_α} $(= \theta_{I_\beta})$ is called a **distinguishing sequence** of the pair (s_i, s_j) iff

$$\theta_{O_\alpha} \neq \theta_{O_\beta} \bullet$$

If the shortest distinguishing sequence of the pair (s_i, s_j) *has a length k, then,* s_i *and* s_j *are said to be* ***k-distinguishable.***

The aim of a homing sequence is to determine the final state of a system observing its outputs. The homing sequence problem was completely solved for sequential machines[4].

I changed this definition because it can happen that output traces are equal for two different execution fragments (see the example in Figure 5), but in that case the final state is the same for both executions (if the output traces are different, the final states can be different, but it is not always the case). Please see if you agree with this definition.

Definition 3.10: Homing sequence

An input trace θ_I^k *of an execution fragment* α_i^k *is a homing sequence iff* $\forall \alpha_i^k, \alpha_j^k \in execs_{Ik}(D)$, *with*

$trace(\alpha_i^k) = (\theta_I^k, \theta_0^{k,i}, w)$, $trace(\alpha_j^k) = (\theta_I^k, \theta_0^{k,j}, w)$, *only one of the following conditions occurs:*

1. $\theta_0^{k,i} \neq \theta_0^{k,j}$.ç

2. $\theta_0^{k,i} = \theta_0^{k,j} \wedge last\left(\alpha_i^{\,k}\right) = last\left(\alpha_j^{\,k}\right).$

It should be clear that only reduced U.B.DEVS have homing sequences, since equivalent states cannot be distinguished. Every reduced U.B.DEVS has a homing sequence.

A synchronizing sequence gives the same final state, regardless of the initial state.

I changed this definition because the quantifiers were incorrectly ordered.

Definition 3.11: Synchronizing sequence

An *input trace* $\theta_l^{\,k}$ *of an execution fragment* $\alpha_i^{\,k}$ *is a synchronizing sequence iff:*

$$\exists! s_l \in S_p \bullet s_l = last(\alpha_i^{\,k}) \; \forall \alpha_i^{\,k} \in execs_{Ik}(D).$$

Having defined the kinds of sequences that are necessary for our test purposes, we can now introduce the different types of test experiments on an U.B.DEVS.

Definition 3.12: Preset Experiment

A preset experiment for an U.B.DEVS is any input event sequence π *of the form* $x_1 x_2 \ldots x_n$ *where* $x_i \in X_{\mathcal{D}} \; \forall i = 1..n$. *The sequence of output events* (*the output trace*) *that* $\mathcal{D}$ *generates in response to* π *is the result of the experiment.*

Definition 3.13: Adaptive Experiment

An adaptive experiment for an U.B.DEVS model $\mathcal{D}$ *is any input event sequence* $\pi = x_1 x_2 \ldots x_n$ *where* $x_i \in X_{\mathcal{D}}\ \forall i = 1..n$ *provided that there exists a function*

$$f: \quad Seq(X_{\mathcal{D}}) \times Y_{\mathcal{D}} \rightarrow X_{\mathcal{D}}$$

such that $\forall i = 2..n \bullet \exists y_i \in Y_{\mathcal{D}} \mid x_i = f(\langle x_0,..,x_{i-1}\rangle, y_i)$.

In other words, the value of the i^{th} input event of an adaptive experiment π depends on all the previous input events and on an (unspecified) output event. The sequence of output events (the output trace) that $\mathcal{D}$ generates in response to π is the result of the experiment.

Since there are many possible definitions for *f*, in order to clearly identify those functions which are of interest for our purposes, we define the concept of a *valid adaptive experiment*.

Definition 3.14: Valid Adaptive Experiment

A valid adaptive experiment $\pi = x_1 x_2 \ldots x_n$ *is an adaptive experiment which the adaptive function f verifies the following property:*

$$x_1 = f(\langle x_0\rangle, y_0) \Leftrightarrow y_0 = \lambda_{\mathcal{D}}(\delta_{ext\mathcal{D}}(s_0, x_0))$$

$$x_i = f(\langle x_0,..,x_{i-1}\rangle, y_{i-1}) \Leftrightarrow y_{i-1} = \lambda_{\mathcal{D}}(\delta_{ext\mathcal{D}}(f(\langle x_0,..,x_{i-2}\rangle, y_{i-2}), x_{i-1})) \qquad \forall i = 2..n$$

In this way, the value of the i^{th} input event of π depends on all the previous input events and the last output event the model has generated. This is equal to saying that the i^{th} input event depends on all the output events that the U.B.DEVS model has generated so far.

Definition 3.15: Distinguishing, Homing, Synchronizing Experiment

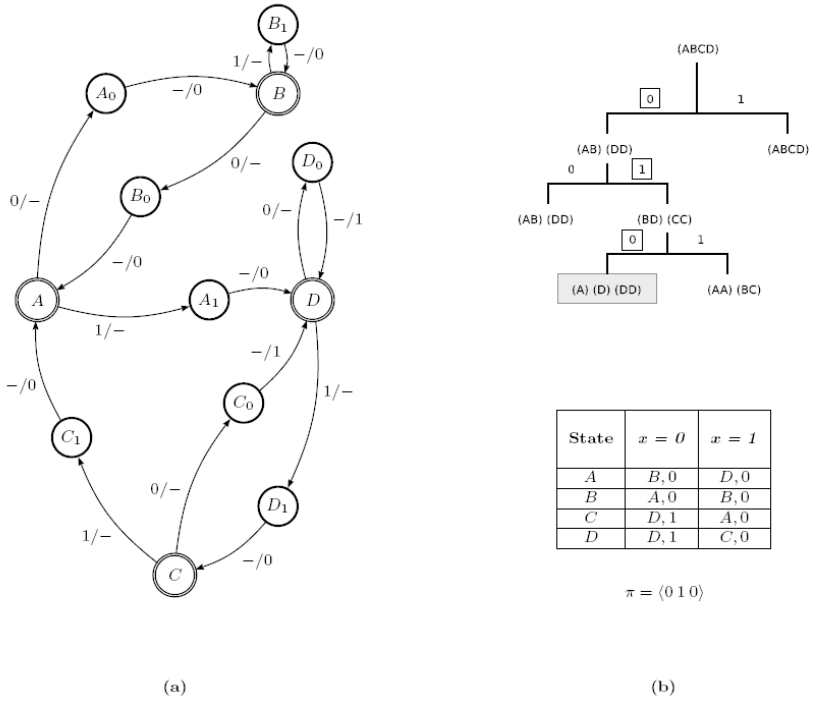
A valid experiment is distinguishing, homing or synchronizing if the input sequence is distinguishing, homing or synchronizing respectively.

3.3 Sequence Finding

In order to perform a *homing experiment* on an U.B.DEVS model, the procedure described in [4] can be applied considering only the passive states.

Theorem 3.2: *A preset homing sequence, whose length is at most* $(n-1)^2$*, exists for every minimal U.B.DEVS model* $\mathcal{D}$*, where n is the number of passive states in* $\mathcal{D}$ [4].

As an example, we give in Figure 5 an U.B.DEVS model and the homing sequence obtained by the procedure[4] considering only the passive states of the model.



State	$x = 0$	$x = 1$
A	$B, 0$	$D, 0$
B	$A, 0$	$B, 0$
C	$D, 1$	$A, 0$
D	$D, 1$	$C, 0$

Figure 5: An U.B.DEVS model (a) with its homing tree, associated transition table, a homing sequence (π) for it (b).

Distinguishing and *synchronizing* sequences can be obtained by the methods presented in [4]. It is straightforward to prove that all the properties and theorems are valid for U.B.DEVS models considering only the passive transition function and the passive output function. In particular, the following result is valid also for U.B.DEVS models:

Theorem 3.3: *If there exists a synchronizing sequence for an U.B.DEVS model* $\mathcal{D}$ *that has n passive states, then its length is at most* $(n-1)^2 n/2$.

Proof: *See* [4]*, p. 458.*

The following result sums up the preceding discussion:

Theorem 3.4: *Let* $\pi = \langle x_1 x_2 ..x_n \rangle$ *be either a synchronizing, homing, preset distinguishing or adaptive distinguishing sequence for a Mealy Machine* $\mathcal{M}$*. Then the sequence* π *is, respectively, either a synchronizing, homing, preset distinguishing or adaptive distinguishing sequence for the U.B.DEVS model* $\mathcal{D}$ *obtained from* $\mathcal{M}$*.*

Up to this point, we have shown it is possible to define concepts within U.B.DEVS theory that are equivalent to concepts from Mealy Machines. In the following section we extend all these results to timed models, considering the Simple-DEVS subset of DEVS models.

4 Simple Bipartite DEVS (S.B.DEVS)

It should be evident that Untimed.DEVS represents a tiny subset of the systems than can be modelled using the DEVS formalism. Then, we need to expand this subset in order to apply fault detection techniques to a wider range of DEVS models. In this first approach, the models considered are Simple.DEVS models (see Definition 2.7):

Definition 4.1: Simple Bipartite DEVS (S.B.DEVS)

A Simple.DEVS model is a S.B.DEVS if and only if

- $S_{\mathcal{D}} = S_{a\mathcal{D}} \cup S_{p\mathcal{D}}$ where $S_{a\mathcal{D}} \cap S_{p\mathcal{D}} = \varnothing$.
- $\delta_{ext\mathcal{D}} : S_{p\mathcal{D}} \times X \to S_{a\mathcal{D}}$
- $\delta_{int\mathcal{D}} : S_{a\mathcal{D}} \to S_{p\mathcal{D}}$
- $\lambda_{\mathcal{D}} : S_{a\mathcal{D}} \to Y_{\mathcal{D}}$
- $ta_{\mathcal{D}} : S_{\mathcal{D}} \to \Re_0^+ \cup \{\infty\}$ where

- $\forall s_i \in S_{p\mathcal{D}} \bullet ta_{\mathcal{D}}(s_i) = \infty$ and
- $\forall s_j \in S_{a\mathcal{D}} \bullet ta_{\mathcal{D}}(s_j) = k, k \in \Re_0^+$.

S.B.DEVS models constitute a subset of the set of Simple.DEVS models, having the following restrictions:

- two consecutive states cannot belong to the same class, the next states of a passive state are active and vice-versa:

  $s_i \in S_a$, $\delta_{int}(s_i) = s_k \Rightarrow s_k \in S_p$ and $s_i \in S_p$, $\forall x_i \in X$, $\delta_{ext}(s_i, x_i) = s_k \Rightarrow s_k \in S_a$
- no external event can occur in an active state :

  $\forall s_i \in S_a$, δ_{ext} is undefined.

Remark: In fact, the "no external event can occur in an active state" hypothesis is the classical hypothesis used in the field of sequential machines: "no input event can occur during a transitory state".

Definition 4.2: Distinguishing sequence

Two passive states s_i *and* s_j *of a S.B.DEVS model* $\mathcal{D}$ *are distinguishable if and only if there exists at least two execution fragments* $\alpha = v_{\alpha_0} x_{\alpha_1} v_{\alpha_1} x_{\alpha_2} v_{\alpha_2} \ldots x_{\alpha_n} v_{\alpha_n}$ *and* $\beta = v_{\beta_0} x_{\beta_1} v_{\beta_1} x_{\beta_2} v_{\beta_2} \ldots x_{\beta_n} v_{\beta_n}$ *of* $\mathcal{D}$ *with* $trace(\alpha) = (\theta_{I_\alpha}, \theta_{O_\alpha}, t_\alpha)$, $trace(\beta) = (\theta_{I_\beta}, \theta_{O_\beta}, t_\beta)$ *where* $v_{\alpha_0}(e) = (s_i, ta(s_i)) \ \forall e \in I_{\alpha_0}$, $v_{\beta_0}(e) = (s_j, ta(s_j)) \ \forall e \in I_{\beta_0}$, $\theta_{I_\alpha} = \theta_{I_\beta}$ *and* $(\theta_{O_\alpha} \neq \theta_{O_\beta})$. *The timed sequence* θ_{I_α} *(and* θ_{I_β}*) is called a distinguishing sequence of the pair* (s_i, s_j). *If there exists for pair* (s_i, s_j) *a distinguishing sequence of length k, then the states in* (s_i, s_j) *are said to be k-distinguishable.*

As we did for the U.B.DEVS models, we introduce the concepts of passive transition function, noted δ_{pass}, and passive output function, noted λ_{pass} for S.B.DEVS.

Definition 4.3: Passive Transition Function

The passive transition function of S.B.DEVS is defined as follows:

$$\forall x_i \in X, s_i \in S_p, \qquad \delta_{pass}(si, xi) = (\delta_{int}(\delta_{ext}(s_i, x_i)), ta(\delta_{int}(s_i)))$$

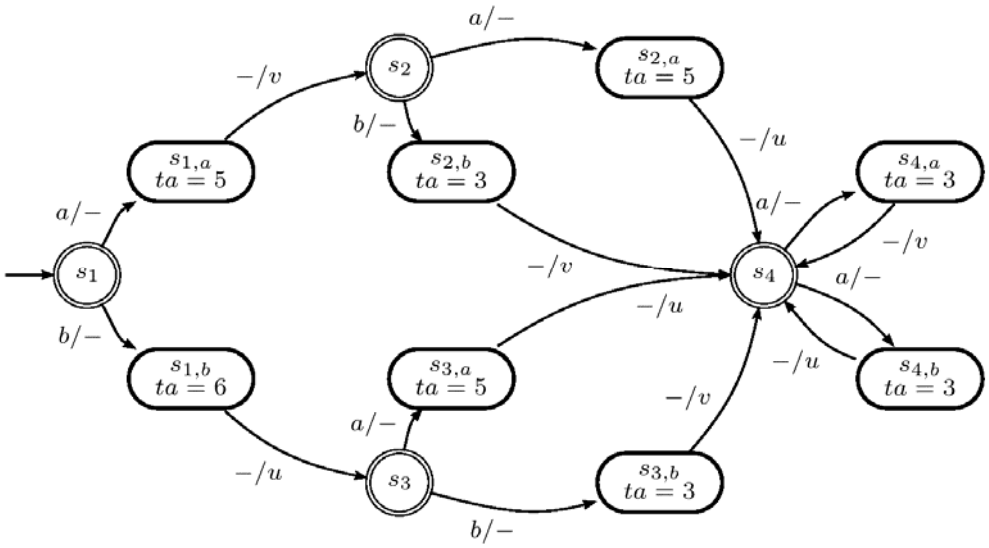
Definition 4.4: Passive Output Function

The passive output function of an U.B.DEVS is defined as follows:

$$\forall x_i \in X, s_i \in S_p, \qquad \lambda_{pass}(s_i, x_i) = \lambda(\delta_{ext}(s_i, x_i)) = y_i \in Y$$

4.1 Minimization procedure for S.B.DEVS models

A state transition table with the passive transition function can be used to represent a S.B.DEVS model, in this case, by adding the value of the lifetime of the next active state (Figure 6).



State	x = a	x = b
S1	s2, 5, v	s3, 6, u
S2	s4, 5, u	s4, 3, v
S3	s4, 5, u	s4, 3, v
S4	s4, 3, v	s4, 3, u

Figure 6: A S.B.DEVS model with its associated transition table.

The minimization procedure defined in [4] is extended in order to take into account the value of $ta(s_{i,x})$ (lifetime of the active state following a passive state s_i that received the input event x). In the first step of this procedure, the set of passive states is partitioned in such a way that each class contains 1-equivalent passive states. We recall that two passive states s_i and s_j are 1-equivalent iff for every input event they produce the same output event ***at the same time***, formally:

$$\forall s_k \in Successor(s_i),\ \exists s_l \in Successor(s_j) \bullet \lambda(s_k) = \lambda(s_l) \wedge ta(s_k) = ta(s_l) \Rightarrow s_i \text{ and } s_j \text{ are } 1-equivalent$$

In the example of Figure 6, the first partition of the state set according to the output values and the lifetime of the considered active states, is:

$$P_1 = \{(s_1), (s_4), (s_2, s_3)\}.$$

Notice that s_1 and s_4 are not in the same class because the lifetime of their next actives states are different, then the timestamps of the corresponding output events are different. These two states cannot be distinguished by the values of the output events but by the occurrence times of these events. On the other hand, s_2 and s_3 are 1-equivalent.

By building the second partition (formed by 2-equivalent states), we conclude that s_2 and s_3 are 2-equivalent, since both transition to s_4 after the first input event (and after its corresponding output event). Moreover, since every input event in s_4 triggers a loop, s_2 and s_3 are k-equivalent $\forall k \geq 3$. Then, we conclude that s_2 and s_3 are two equivalent states, then we can choose s_3 to represent the equivalence class. Figure 7 gives the reduced form of the S.B.DEVS example.

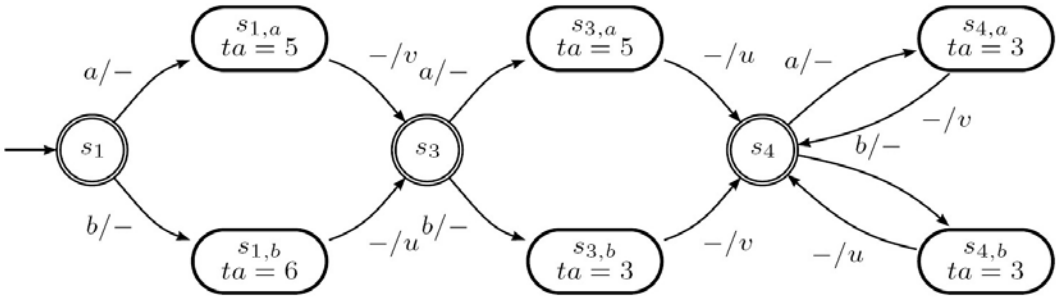


Figure 7: Resulting minimal S.B.DEVS model.

4.2 Fault detection techniques on S.B.DEVS models

In order to respect the hypothesis that no external event can occur in an active state, all the input traces considered are slow timed input traces.

The definitions of homing and synchronizing sequences must be extended taking into account the timed aspects of the formalism.

In the following, we denote with $execs_{Ik}(\mathcal{D})$ *the set of slow timed and finite executions of* $\mathcal{D}$ *with the same timed input trace* θ_I^k.

Definition 4.5 : Homing sequence

A timed input trace θ_I^k *of a slow timed execution fragment* $\alpha_i^k = v_{i0}e_{i0}v_{i1}e_{i1}.....e_{in}v_{in}$ *is a homing sequence iff* $\forall \alpha_i^k, \alpha_j^k \in execs_{Ik}(D)$, *with* $trace\left(\alpha_i^k\right) = \left(\theta_I^k, \theta_0^{k,i}, w\right), trace\left(\alpha_j^k\right) = \left(\theta_I^k, \theta_0^{k,j}, w\right)$, *only one of the following conditions occurs:*

3. $\theta_0^{k,i} \neq \theta_0^{k,j}$.

4. $\theta_0^{k,i} = \theta_0^{k,j} \wedge last\left(\alpha_i^k\right) = last\left(\alpha_j^k\right)$.

Definition 4.6: Synchronizing sequence

A t*imed input trace* θ_I^k *of a slow execution fragment* $\alpha_i^k = v_{i0}e_{i0}v_{i1}e_{i1}.....e_{in}v_{in}$ *is a synchronizing sequence iff:*

$$\exists! s_I \in S_p \mid (s_I, 0) = v_{in}(Inf(I_{in})) \quad \forall \alpha_i^k \in execs_{Ik}(D).$$

4.3 Experiments on S.B.DEVS models

The definition of a preset experiment (PX for short) for S.B.DEVS models is adapted for handling the timed aspects of such models. We introduce the concept of a timed relative input trace in order to refer not to the absolute time of each event, but to its relative time with respect to the previous input event. That is, 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, so so it is equivalent to talk about either θ_I or $rel(\theta_I)$.

I need to insist on this definition (it was in rev. 7 but not in rev. 8) because in all the following sections, w is the waiting time after the last input event. As a consequence, we either have to change all the rest of the text, figures and tables so as to give w the same meaning (for example, in section 4.4, the sequence should be $(\pi,28.3)$ instead of $(\pi,9.1)$), or change this definition. Please inform me to change the figures, should you choose w to mean the length of the experiment.

Definition 4.7: Preset Experiment

A preset experiment for a S.B.DEVS model $\mathcal{D}$ *is any pair* (θ_I,w) *where* $\theta_I \in (X_{\mathcal{D}} \times \Re_0^+)^*$ *is a timed input trace of the form* $\langle (x_0,t_0),..,(x_n,t_n)\rangle$ *and* $w \in \Re_0^+$ *is the time during which the experiment must go on after the last event in the input trace. The output trace that* $\mathcal{D}$ *generates in response to* θ_I *after l units of time is the result of the experiment, and l is the length of the experiment, where* $l = \sum_{i=0}^{n} t_i + w$.

It is straightforward to see that not all sequences that belong to $(X_{\mathcal{D}} \times \Re_0^+)^*$ can be applied to a given S.B.DEVS model. The subset of sequences that will be accepted constitute valid experiments.

Definition 4.8: Valid Preset Experiment

A valid preset experiment for a S.B.DEVS model $\mathcal{D}$ *is any pair* (θ_I, w), *where* $\theta_I \in (X_{\mathcal{D}} \times \Re_0^+)^*$ *is a* ***slow timed input trace*** *and* $w > t = max\{ta(s) \mid s \in S_{a\mathcal{D}}\}$.

Definition 4.9: Adaptive Experiment

An adaptive experiment for a S.B.DEVS model $\mathcal{D}$ *is any pair* (θ_I, w) , *where* $\theta_I \in (X_{\mathcal{D}} \times \Re_0^+)^*$ *is a slow timed relative input sequence of the form* $\langle (x_0, t_0), .., (x_n, t_n) \rangle$ *and* $w \in \Re_0^+$, *provided that there exists a function*

$$f : Seq(X_{\mathcal{D}}) \times Y_{\mathcal{D}} \to X_{\mathcal{D}}$$

$$such\ that\ \forall i = 2..n \bullet \exists y_i \in Y_{\mathcal{D}} \mid x_i = f(\langle x_0, .., x_{i-1} \rangle, y_i)$$

That is to say that the value of the i^{th} input event in θ_I depends on all the previous input events and on an (unspecified) output event. The sequence of output events (the output trace) generated by $\mathcal{D}$ in response to θ_I is the result of the experiment.

Since there are many possible definitions for *f*, in order to clearly identify those functions which are of interest for our test purposes, we define the concept of a *valid adaptive experiment*:

Definition 4.10: Valid Adaptive Experiment

An adaptive experiment (θ_I, w) *for a S.B.DEVS model* $\mathcal{D}$ *is a valid experiment iff* θ_I *is slow on* $\mathcal{D}$, $w \geq max\left\{ta_{\mathcal{D}}(s_{a_i}) \mid s_{a_i} \in S_{a\mathcal{D}}\right\}$, *and the function f of this experiment satisfies the following property:*

$$x_1 = f(\langle x_0 \rangle, y_0) \Leftrightarrow y_0 = \lambda_{pass}(s_0, x_0)$$

$$x_i = f(\langle x_0, .., x_{i-1} \rangle, y_{i-1}) \Leftrightarrow y_{i-1} = \lambda_{pass}(f(\langle x_0, .., x_{i-2} \rangle, y_{i-2}), x_{i-1}) \qquad \forall i = 2..n$$

In other words the value of the i^{th} input event in θ_I depends on all the previous input events and the last output event the model has generated. This is equal to saying that the i^{th} input event depends on all the output events that the S.B.DEVS model has generated so far.

Definition 4.11: Distinguishing, Homing, Synchronizing Experiments on S.B.DEVS models

A valid experiment (θ_I, w) *on a S.B.DEVS model* $\mathcal{D}$ *is distinguishing, homing or synchronizing iff* θ_I *is a distinguishing, homing or synchronizing sequence for* $\mathcal{D}$, *respectively.*

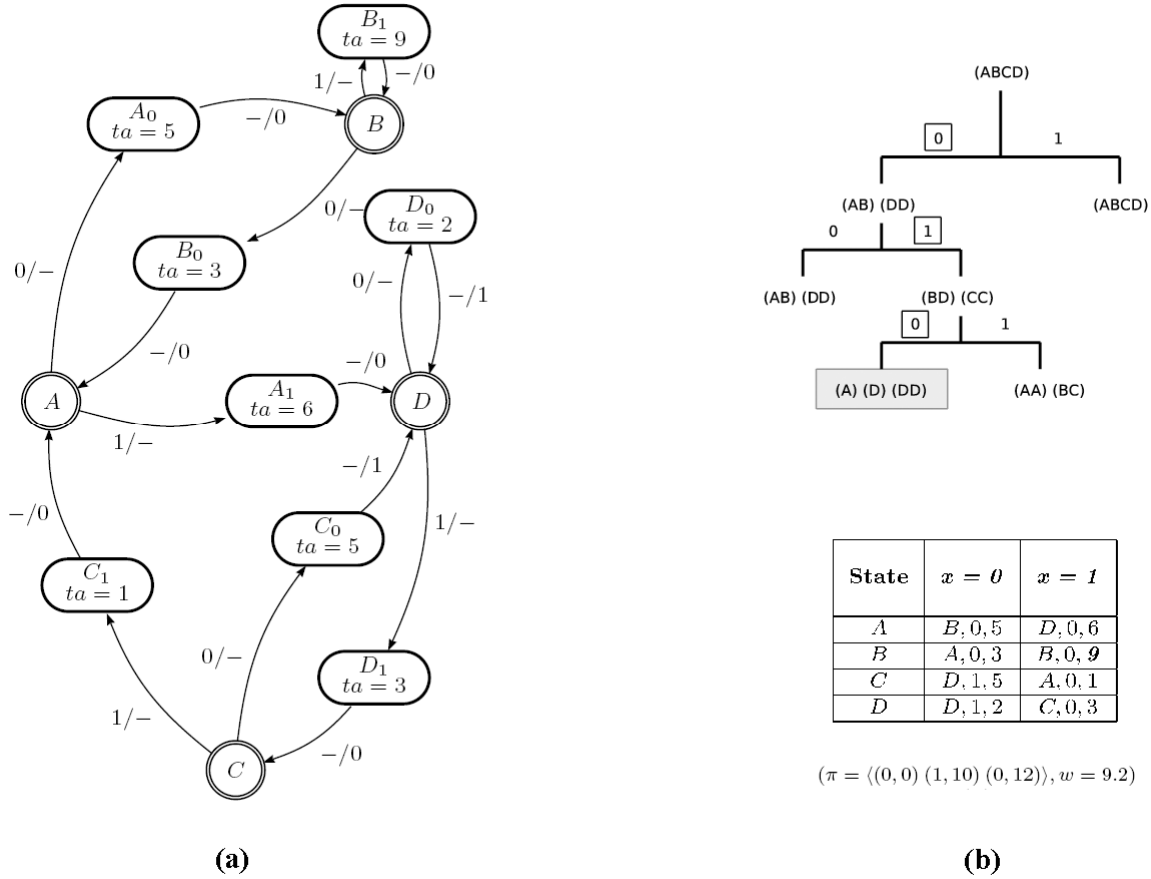
4.4 Sequence Finding

In order to perform a *homing experiment* on a S.B.DEVS model, the procedure described in [4] can be utilized in a straightforward way, by considering the passive transition function and the concept of slow timed execution fragments.

For a minimal S.B.DEVS, a preset homing sequence is built, without timing considerations, using the passive transition function. Then, to each input event x_k of this untimed sequence we provide a ***relative*** time stamp t_k such that $t_k > t = max\{ta(s) \mid s \in S_{a\mathcal{D}}\}$ in order to obtain a slow timed

relative input trace. Finally, choosing a waiting time *w* such that $w > t$, *the pair* (θ_I, w) *defines a valid preset experiment.*

Example: Let us consider the S.B.DEVS represented in Figure 8. Using the method presented in [4] we obtain the following untimed homing sequence: $\langle 0,1,0 \rangle$. Notice that $t = max\{ta(s) \mid s \in S_{a\mathcal{D}}\} = 9$; a possible timed relative homing sequence for this S.B.DEVS is then: $\pi = \langle (0, t=0),\ (1,\ t=10),\ (0,\ t=9.2) \rangle$, and $(\pi, 9.1)$ is a valid experiment for this model.



State	$x = 0$	$x = 1$
A	$B, 0, 5$	$D, 0, 6$
B	$A, 0, 3$	$B, 0, 9$
C	$D, 1, 5$	$A, 0, 1$
D	$D, 1, 2$	$C, 0, 3$

$(\pi = \langle (0,0)\ (1,10)\ (0,12) \rangle, w = 9.2)$

Figure 8: S.B.DEVS model (a) with its homing tree, associated transition table, and one possible homing sequence (π) for it (b).

Theorem 4.1: *A preset homing sequence, whose length is at most* $(n-1)^2$*, exists for every minimal S.B.DEVS model* $\mathcal{D}$*, where n is the number of passive states in* $\mathcal{D}$.

Given the previous considerations on the timed nature of sequences, both *distinguishing* and *synchronizing* sequences can be obtained using the methods and algorithms described in [4], and afterwards adjoining the time stamps to the events as we did with homing experiments. It is straightforward to prove that all the properties and theorems in [4] are valid for S.B.DEVS models. In particular, the following result is valid:

Theorem 4.2: *If there exists a synchronizing sequence for a S.B.DEVS model $\mathcal{D}$ that has n passive states, then its length is at most* $(n-1)^2 n/2$.

Proof: See [4], p. 458.

The following result sums up the preceding discussion:

Theorem 4.3: *Let* $\pi = \langle x_1 x_2 ..x_n \rangle$ *be either a synchronizing, homing, preset distinguishing or adaptive distinguishing sequence for a Mealy Machine $\mathcal{M}$. Then the sequence* $\upsilon = \langle (x_1,t_1)(x_2,t_2)..(x_n,t_n) \rangle$ *obtained from π is, respectively, either a synchronizing, homing, preset distinguishing or adaptive distinguishing sequence for the S.B.DEVS model $\mathcal{D}$ with the same input-output behaviour than $\mathcal{M}$, if* $t = max\{ta_{\mathcal{D}}(p) | p \in S_{a\mathcal{D}}\} < t_i, \forall i = 1..n$.

4.5 Testing procedure

4.6 Now, we need to define a testing procedure to test U.B.DEVS and S.B.DEVS models. For the first type of models the procedure is straightforward since the inputting process is

independent of time.

For S.B.DEVS models, the testing procedure can be formally represented by a tester[5,6,10]. For the sake of constancy, we choose to specify the tester by a DEVS model. The implementation of the S.B.DEVS under test receives the test input sequence from the tester output tester at specific times. The general scheme of the coupling between a tester and the model under test is given in Figure 9.

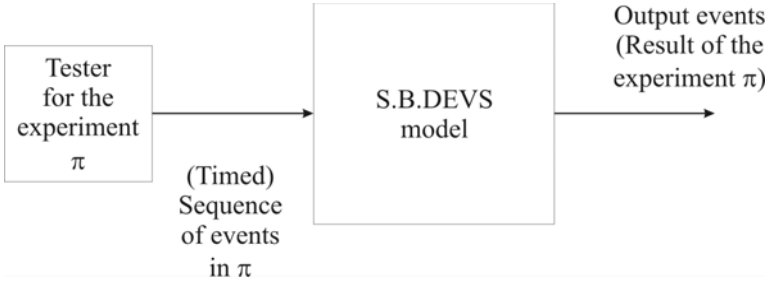


Figure 9: Coupling scheme of a S.B.DEVS model and a valid tester for it.

The tester for a given PX is defined as follows:

Definition 4.12: DEVS tester model for preset experiments

Given a PX (π, w) *where* $\pi = (x_0,t_0),(x_1,t_1),\ldots,(x_n,t_n)$, *its associated tester is the DEVS model* $\mathcal{T} = < X_T, Y_T, S_T, \delta_{int_T}, \delta_{ext_T}, \lambda_T, ta_T >$, *with:*

- $X_T = \{Reset\}$ (*Event that restores the tester to its initial state*)
- $Y_T = x_0, x_1, \ldots, x_n$
- $S_T = s_0, s_1, \ldots, s_n \cup s_{STOP}$
- $\delta_{ext_T}(s_i, e, x) = s_0$ (*Restores the tester to its initial state*)

- $\delta_{int_T}(s_i) = \begin{cases} s_{i+1} & \text{if} \quad i < n \\ s_{STOP} & \text{if} \quad i = n \vee s_i = s_{STOP} \end{cases}$

- $ta_T(s_i) = \begin{cases} 0 & \text{if} \quad i = 0 \\ t_i & \text{if} \quad i = 1..n \wedge s_i \neq s_{STOP} \\ \infty & \text{if} \quad s_i = s_{STOP} \end{cases}$

- $\lambda_T(s_i) = \begin{cases} x_i & \text{if} \quad s_i \neq s_{STOP} \\ 0 & \text{if} \quad s_i = s_{STOP} \end{cases}$ (*This case never happens as* $ta_T(s_{STOP}) = \infty$)

Example: the tester that implements the experiment ($\pi = (a,0)(b,2)(c,1)$, $w = 4$) is represented by the state graph in Figure 10:

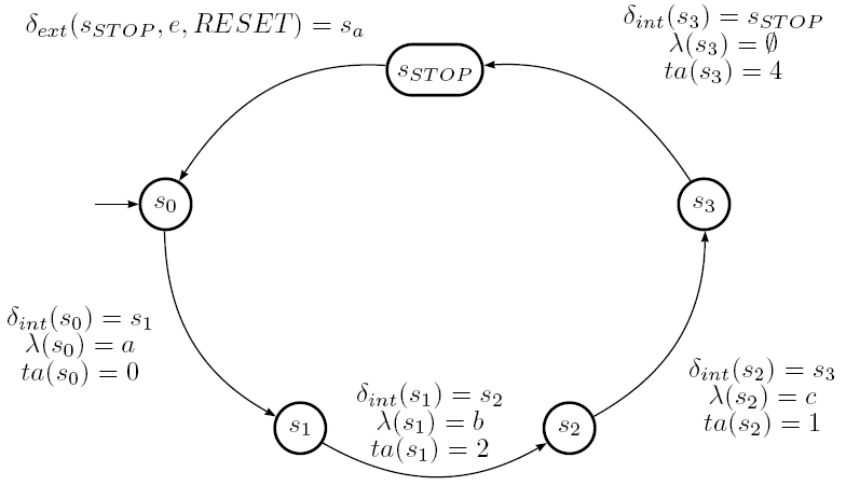


Figure 10: Tester for the preset experiment (PX) (π = (*a*,0)(*b*,2)(*c*,1), *w* = 4).

If we take into account the fact that after the tester sends the output event x_i, the S.B.DEVS model under test will transition up to the passive state s_{i+1}, then it is assured that the S.B.DEVS model will remain in the same state until the tester issues the output event x_{i+1} in case it exists.

Example: In order to implement a ***valid adaptive experiment*** for a S.B.DEVS model, the basic

idea is to define a tester which represents the decision tree associated with the adaptive experiment, which is an extended version of the one given in ([4], p.461) taking into account the elapsed time in the states. In the tester, it is required to have one active state for each output event that the experiment will issue. Additionally, each of these states will receive an external event that represents the response of the model under test. That is, it will be able to receive any of the possible output events that the S.B.DEVS model will generate (in order to do this, the output of the tested model needs to be connected to the tester's input). Depending on the value of the received event, the tester will transition to one of the active states that represent the consequential uncertainties. Finally, all the leaf nodes have to be represented as passive states which only accept the RESET event in order to reinitialize the experiment. For a complete specification of the tester, all unacceptable behaviour of the tested model (that is, if the tested model outputs an unexpected event or an expected event at an unexpected time) must force the tester to transition to an error state. We omit this state in the following example so as to make it easier to read, but it should be always added in the final specification. The following figure shows a concrete example of an adaptive tester:

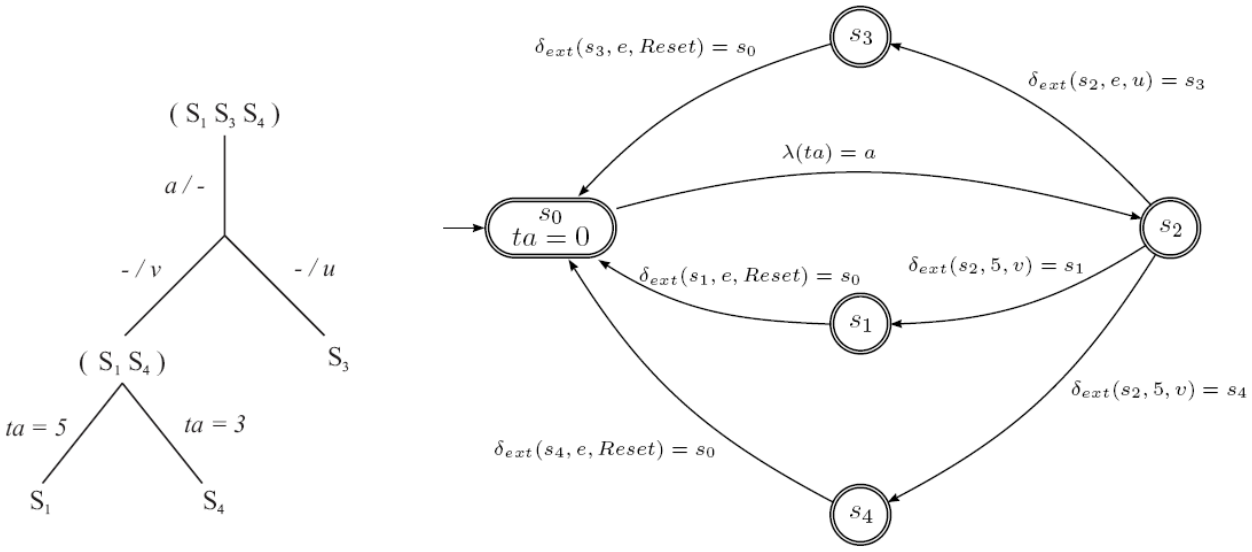


Figure 11: Sample adaptive distinguishing experiment and corresponding DEVS tester model for the S.B.DEVS model in Figure 7.

5 Conclusions And Future Work

The definitions and methods presented in this paper allow for a proper identification of the states in useful subsets of DEVS models. All the states of a Mealy machine are steady states; we have introduced Untimed.DEVS to allow a clean specification of transitory states in untimed models, since they represent those models which can be properly specified in Mealy machines, and thus providing a clear way to port Mealy machines to a timed formalism such as DEVS. Additionally, U.B.DEVS models represent the smallest useful subset of timed models whose states can be properly identified and fault checked. This subset constitutes the first step towards a formal theory of fault detection techniques for the more general DEVS models.

Aside from the definition of the before mentioned subsets, the importance of formally defining the concepts of distinguishing, homing and synchronizing sequences lies in the possibility to check random sequences for a given DEVS model in order to determine whether they belong to

any of the three relevant categories. A method to obtain such sequences could then be designed so as to provide a mechanical, automatic means to find relevant sequences of timed input events. The process of obtaining new sequences can be also automated since it is straightforward to find the highest value of the time advance function *ta* in a model, and then design the proper tester for every possible slow timed input sequence that the model accepts. This ensures the plausibility to automate a stage of the conformance testing procedure.

Future work involves expanding the U.B.DEVS subset, possibly up to clearly stating the broadest subset of DEVS models that can be fault checked with traditional techniques, as well as targeting the problem of fault detection in coupled DEVS models.

6 References

1. Alur, Rajeev and Dill, David, Theoretical Computer Science **126**(2), 183 (1994).
Ref Type: Journal

2. H.P.Dacharry and N.Giambiasi, *Formal Verification with Timed Automata and DEVS Models: a case study*, Rosario, Argentina, 2005).

3. H.P.Dacharry and N.Giambiasi, *DEVS based timed hierarchy of formalisms*, 2007).

4. Z.Kohavi, *Switching and Finite Automata Theory: Computer Science Series*, Richard W. Hamming and Edward A. Feigenbaum (McGraw-Hill Higher Education, 1980).

5. M.Krichen and S.Tripakis, *State identification problems for timed automata*, *LNCS*, (Springer, 2005).

6. Krichen, Moez and Tripakis, Stavros, Verimag, 2005.

7. D.Lee and M.Yannakakis, *Principles and Methods of Testing Finite State Machines - A Survey*, 1996).

8. E.Moore, *Gedanken Experiments on Sequential Machines*,(Princeton U., 1956).

9. Springintveld, Jan, Vaandrager, Frits, and D'Argenio, Pedro R., Theoretical Computer Science **254**(1--2), (2001).
Ref Type: Journal

10. Tripakis, Stavros and Yovine, Sergio, Formal Methods in System Design **18**(1), (2001).
Ref Type: Journal

11. B.P.Zeigler and H.Praehofer, *Theory of Modeling and Simulation*,(Academic Press, 2000).

12. B.P.Zeigler, H.Praehofer, and T.G.Kim, *Theory of Modeling and Simulation, Second Edition*,(Academic Press, 2000).

7 Contents