Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 195 (2008) 21–40
www.elsevier.com/locate/entcs

Towards Deriving Test Sequences by Model Checking

Adilson Luiz Bonifácio[1] Arnaldo Vieira Moura[2]

Computing Institute, University of Campinas
P.O. 6176 – Campinas – Brazil – 13081-970

Adenilso da Silva Simão[3] José Carlos Maldonado[4]

Mathematic Science and Computing Institute, University of São Paulo
P.O. 668 – São Carlos – Brazil – 13560-970

Abstract

Model-based testing automatically generates test cases from a model describing the behavior of the system under test. Although there exist several model-based formal testing methods, they usually do not address time constraints, mainly due to the fact that some supporting formalisms do not allow a suitable representation of time. In this paper, we consider such constraints in a framework of Timed Extended Finite State Machines (TEFSMs), which augment the Extended Finite State Machine (EFSM) model by including a notion of explicit and implicit time advancement. We use this extension to address conformance testing by reducing the confirming configuration problem to the problem of finding a path in a TEFSM product.

Keywords: Model Checking, Timed EFSM, Conformance testing, Suspicious Configuration.

1 Introduction

Model-based testing comprises the automatic generation of efficient test cases using models of system requirements, usually based on formally specified system functionalities. It involves the (i) construction of a suitable formal model, (ii) derivation of test inputs, (iii) calculation of test outputs, (iv) execution of test inputs over implementations, (v) comparison of the results from the calculated test outputs and the implementation executions, and (vi) decision of whether the testing should be stopped. All these tasks are tightly related to each other. For instance, the

[1] Email: adilson@ic.unicamp.br
[2] Email: arnaldo@ic.unicamp.br
[3] Email: adenilso@icmc.usp.br
[4] Email: jcmaldon@icmc.usp.br

doi:10.1016/j.entcs.2007.08.025

way the model is written impacts on how test inputs can be generated. Moreover, the decision of whether the implementation has already been tested enough depends on one's ability to determine how many undiscovered faults may remain in it. Usually the purpose of testing is not to demonstrate that the implementation is equivalent to its specification, since this goal is infeasible for most practical applications. Instead, this ideal equivalence is relaxed into a conformance relation [13,15]. The so-called conformance testing aims at demonstrating that the implementation behavior conforms (in some sense) to the behavior dictated by the specification [29].

The problem of generating test cases for conformance testing based on Finite State Machines (FSMs) has already been investigated [7,21,28,8,14,12]. However, there are many situations in which the modeling of the system as a FSM is cumbersome, due to the state explosion problem, or even impossible, due to the fact that there are some relevant aspects that can not be properly expressed, e.g., the passage of time. Some extensions to the FSM model have been proposed in order to overcome these problems [33,6,1]. Other extensions incorporate notions like context variables and input/output parameters, allowing the succinct representation of many different configurations [27]. Still others incorporate notions of time, allowing the model to capture the evolution of time [24,4,36].

An Extended Finite State Machine (EFSM) can be thought of as a folded FSM [27]. Given an EFSM, and assuming that domains are finite, it is possible to unfold it into a pure FSM by expanding the values of its parameters and variables. The resulting FSM can be used with FSM-based methods for test derivation with complete fault coverage, which means that all fault possibilities can be exhausted. Nonetheless, in most practical situations, this approach is unfeasible, mainly due to the state explosion effect [22,27].

Time plays an important role in determining the acceptability of system behavior in many system categories since not only the input/output relationship can be relevant, but also the period of time when those events occur may be important. In such cases, it is mandatory to be able to represent time constraints of the system, and to test whether a given implementation conforms to these constraints. There are some formalisms that allow the representation of various time related concepts, such as Timed Petri Nets [19] and Timed Automata [2,1,32,11]. Nonetheless, there are few, if any, methods that allow a satisfactory derivation of adequate test cases from those models.

We are interested in model-based methods for testing systems with time constraints. In particular, we are addressing the problem posed in tasks (i)-(iii) alluded to above, namely the construction of an adequate formalism for modeling systems and the automatic generation of test cases, as well as the determination of the expected outputs. These tasks are closely related, and should be considered together.

To this end, we define Timed EFSMs [5], or TEFSMs, by including the notion of explicit and implicit time advancement in the EFSM formalism. Then, we can adapt some well-established results, derived for FSMs and EFSMs, to the context of systems that require time constraints. In particular, we address the problem of configuration confirmation for TEFSMs in the same vein as done by Petrenko

et al. for EFSMs [27]. In that work, it is shown how the problem of configuration confirmation for EFSMs can be reduced to the problem of finding a path in an EFSM product. By defining a property that states when no such a path exists, model-checking techniques can be used to generate a confirming sequence. We show how the notion of product machines and confirming sequences can be applied to the extended formalism of TEFSMs. Given a configuration and a set of suspicious configurations, a confirming sequence is a sequence of (parameterized) inputs that allows us to distinguish the given configuration from suspicious configurations by comparing outputs and, possibly, observing the time indicated in each of the outputs. Finding a confirming sequence can also be seen as an extension of the state identification problem [20,16].

This paper is organized as follows. In Section 2 we present the concepts of EFSMs and Extended Timed Transition Systems [7]. In Section 3 we introduce the Timed Extended FSMs. The product of TEFSMs is presented in Section 4. In Section 5 we describe how the TEFSM product can be used in a model-checking set-up, and illustrate this process with a simple example in Section 6. Finally, in Section 7, we draw some concluding remarks and indicate possible directions for future research.

2 Basic Formal Concepts

In this section, we give a brief overview of the formal concepts that are involved in this work. First, we present EFSMs which are used to specify system requirements. Next, important aspects of extended timed transition systems are introduced.

2.1 Extended FSM Model

An EFSM is an extension of a conventional FSM. In contrast to FSMs, in the EFSM model we have to consider other items [27], such as input and output parameters, and context variables. Also, update and output functions, as well as predicates are defined over context variables and input parameters.

Let X and Y be finite sets of input and output symbols. Let R be a finite set of parameter symbols. For $z \in X \cup Y$, we denote by $R_z \subseteq R$ the set of parameters associated with z. Also D_z denotes the set of parameter valuations associated with z. An element of D_z maps R_z to some valuation domain. Similarly, let V be a finite set of context variable names, with D_V denoting a set of valuations for V. At this point, there is no need to further specify the valuation domains. An EFSM M over X, Y, R, V and the associated valuation domains is a tuple (S, T, s_0, λ_0), where S and T are finite sets of states and transitions, respectively, $s_0 \in S$ is the initial state, and λ_0 is an initial context variable valuation. Each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where:

- $s, s' \in S$ are the source and the target states of the transition, respectively;
- $x \in X$ is the input symbol of the transition;
- $y \in Y$ is the output symbol of the transition;

- P, op and up are functions defined over valuations of the input parameters and context variables V, thus:
  - $P : D_x \times D_V \rightarrow \{True, False\}$ is the predicate of the transition;
  - $op : D_x \times D_V \rightarrow D_y$ is the output parameter function of the transition;
  - $up : D_x \times D_V \rightarrow D_V$ is the context update function of the transition.

Given an input x and the set of input parameter valuations D_x, a parameterized input is a pair (x, p_x), where $p_x \in D_x$. The parameterized outputs are defined in a similar way. A configuration of M is a pair $(s, \lambda) \in S \times D_V$, where s is a state and λ is a context variable valuation. A transition (s, x, P, op, y, up, s') is enabled for a configuration (s, λ) and parameterized input (x, p_x) if $P(p_x, \lambda)$ evaluates to true.

The machine starts from the initial configuration and operates as follows. Upon receiving an input along with the corresponding parameter valuation, and computes the predicates that are satisfied for the current configuration. From among the presently enabled transitions one will fire. By executing the chosen transition, the machine produces an output along with an output parameter valuation using of the output parameter function. The latter is computed by the output parameter valuation. The machine updates the current context variable valuation according to the context update function, and moves from the source to the target state of the transition.

An EFSM, furthermore, is considered to be:

- Predicate complete: for each pair $(s, x) \in S \times X$, every element in $D_x \times D_V$ evaluates at least one predicate to true among the set of all predicates guarding transitions leaving s with input x;
- Input complete: for each pair $(s, x) \in S \times X$, there exists at least one transition leaving state s with input x;
- Deterministic: any two transitions leaving the same state and with the same input have mutually exclusive predicates;
- Observable: for each state s and each input x, every outgoing transition from s on x has a distinct output symbol.

2.2 Extended Timed Transition Systems

We can extend the original *timed transition system* (TTS) notion of [7] by associating a set of clocks and invariant conditions with each state. All clocks in the model increase in an uniform way, according to a global time frame [1,2], and the corresponding invariant condition must hold in the current state of the model.

First, we say how clocks behave during system evolution [1]. Let C be the set of clock names (or clocks, for short), $\Phi(C)$ is the set of clock constraints δ in the form,

$$\delta := c \leq \tau \mid \tau \leq c \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where c is a clock and $\tau \in \mathbb{Q}$ [5] is a time instant. A clock interpretation, ν, is a

[5] $\mathbb{Q}$ is the set of rationals and $\mathbb{Q}^{>0}$ is the set of positive rationals.

mapping from C to $\mathbb{Q}$. The set of clock interpretations is denoted by $[C \mapsto \mathbb{Q}]$. An interpretation ν over C satisfies $\delta \in \Phi(C)$, written $\nu \vDash \delta$, iff δ evaluates to true when each clock c is substituted by $\nu(c)$ in δ.

Let $\nu \in [C \mapsto \mathbb{Q}]$ be a clock interpretation. For $\tau \in \mathbb{Q}$, we define the clock interpretation $\nu + \tau$, which maps each clock c to the value $\nu(c) + \tau$. Also, for $K \subseteq C$, $[K \mapsto \tau]\nu$ is the clock interpretation that assigns $\tau \in \mathbb{Q}$ to each clock $c \in K$ and agrees with ν on the rest of the clocks.

An Extended TTS (ETTS) is given by a tuple $(S, s_0, X, C, Inv, \longrightarrow)$, where S is a finite set of states, $s_0 \in S$ is the initial state, X is a finite set of events, C is a finite set of clocks, $Inv : S \to \Phi(C)$ maps states to invariant conditions, and $\longrightarrow$ is a transition relation, where $\longrightarrow \subseteq (S \times X \times 2^C \times \Phi(C) \times S)$. A configuration is given by a pair (s, ν), where s is a state and ν is a clock interpretation. The initial configuration is given by (s_0, ν_0), where $\nu_0(c) = 0$, for all $c \in C$, is the initial clock interpretation, and $\nu_0 \vDash Inv(s_0)$. Given a configuration (s, ν), a transition (s, x, K, δ, s') indicates that from state s, receiving the input event x, and provided that ν satisfies δ, the system may move to state s', resetting all the clocks in K to zero. The ETTS always starts in the initial configuration (s_0, ν_0), and with the (global) time set to zero.

A time sequence is a sequence $\bar{\tau} = \tau_0\tau_1\tau_2 \ldots$, where $\tau_i \in \mathbb{Q}$, $i \geq 0$, $\tau_0 = 0$, and $\tau_i \geq \tau_{i-1}$, $i \geq 1$. A timed sequence is a pair $(\bar{x}, \bar{\tau})$, where $\bar{\tau}$ is a time sequence and $\bar{x} = x_0x_1x_2 \ldots$ is a sequence of input symbols. The intuitive idea is that the symbol x_i occurs at time τ_i. Given two configurations, (s_1, ν_1) and (s_2, ν_2), a time delay $\tau \geq 0$ and an input x, we say that (s_2, ν_2) evolves from (s_1, ν_1) over τ and x, denoted by $(s_1, \nu_1) \xrightarrow[\tau]{x} (s_2, \nu_2)$, iff there is a transition (s_1, x, K, δ, s_2) such that:

(i) $\nu_1 + \eta \vDash Inv(s_1)$ for all $0 \leq \eta \leq \tau$,

(ii) $\nu_1 + \tau \vDash \delta$,

(iii) $\nu_2 = [K \mapsto 0](\nu_1 + \tau)$, and

(iv) $\nu_2 \vDash Inv(s_2)$.

A sequence of configurations $\bar{\gamma} = \gamma_0\gamma_1\gamma_2 \ldots$ is a run of M iff γ_0 is the initial configuration of M, and there is a timed input $(\bar{x}, \bar{\tau})$ such that

$$\gamma_{i-1} \xrightarrow[\theta_i]{x_i} \gamma_i, \text{ where } \theta_i = \tau_i - \tau_{i-1},\ i \geq 1.$$

In this case, we say that $\bar{\gamma}$ is a run of M over $(\bar{x}, \bar{\tau})$ from γ_0.

Note that, in a timed sequence $(\bar{x}, \bar{\tau})$, time evolves by $(\tau_i - \tau_{i-1})$ units from the moment when x_{i-1} occurred until x_i occurs (for $i > 1$). Intuitively a run captures the system evolution, as follows:

(i) it starts at state s_0, with all clocks set to zero;

(ii) time evolves by $\tau_1 - \tau_0 = \tau_1$ units;

(iii) at instant τ_1 the system changes to state s_1 on input x_1 while resetting clocks in K_1 to zero;

(iv) time evolves by another $\tau_2 - \tau_1$ units;

(v) at instant $\tau_1 + (\tau_2 - \tau_1) = \tau_2$ the system changes to state s_2 on input x_2 while resetting clocks in K_2 to zero;

(vi) and so on.

We can see that:

- a change of state can only occur when the transition (s, x, K, δ, s') is enabled, i.e., when δ is satisfied in the present configuration;
- clocks can be reset to zero in any transition;
- any clock reading is the elapsed time since the last instant it was reset to zero; and
- all clocks increase uniformly according to a global time frame.

3 The Timed EFSM model

In the previous sections, we have presented two formalisms: EFSMs and ETTSs. While EFSMs capture the relationships between inputs, outputs and context variables, ETTSs offer a treatment of time evolution and its constraints. We observe that there are several methods and techniques for deriving tests from (E)FSM models (e.g., [27,17,8,26]). However, the derivation of test cases from (E)TTSs is less established, although some works have considered it (e.g., [30,7,18]). It is worth combining both ETTSs and EFSMs formalisms in order to benefit from the power of both models in terms of expressiveness. This section redefines the EFSM model in order to capture real-time. We use the ETTS definition as inspiration for this purpose.

3.1 Creating a TEFSM model from an ETTS and an EFSM model

Let X be a finite set of inputs, Y be a finite set of outputs, C be a finite set of clocks, R be a finite set of parameters, and V be a finite set of context variables. A Timed Extended Finite State Machine, or TEFSM, M over X, Y, R, V, C, and the associated valuation domains is a tuple $(S, T, Inv, s_0, \nu_0, \lambda_0)$, where S and T are finite sets of states and transitions, respectively, Inv is a finite set of invariant conditions associated with states and, $s_0 \in S$ is the initial state, $\nu_0 = [C \mapsto 0]$ is the initial clock interpretation and λ_0 is an initial context variable valuation. In the TEFSM model: (i) the dynamic behavior is given by clocks and their resetting, as in the ETTS model; and (ii) the data and control flow are given by parameters and context variables, as in the EFSM model. A transition $t \in T$ is expressed by a tuple $(s, x, Q, K, op, y, up, s')$, where:

- s, x, s' and K are as defined in the ETTS formalism; see Section 2.2;
- op, y, and up are as defined in the EFSM formalism; see Section 2.1;
- $Q : D_x \times [C \mapsto \mathbb{Q}] \times D_V \rightarrow \{True, False\}$ is the predicate of the transition.

It can be seen that the TEFSM model comprises the EFSM formalism. That is, given a EFSM M over X, Y, R, V and some valuation domains, as defined in Section 2.1, we can construct a TEFSM model $\hat{M}$ over the same sets X, Y, R, V and the corresponding domains, by letting the clock set C be simply $\{c\}$. For each transition $t = (s, x, P, op, y, up, s')$ in M, we define a transition $\hat{t} = (s, x, Q, K, op, y, up, s')$ in $\hat{M}$ by letting $Q(p_x, \nu, \lambda) = P(p_x, \lambda)$, for any (p_x, ν, λ) in $D_x \times [C \mapsto \mathbb{Q}] \times D_V$. We also let $K = \emptyset$. Clearly, for any $p_x \in D_x$, $\lambda_v \in D_V$ and any clock interpretation $\nu \in [C \mapsto \mathbb{Q}]$, we have that $Q(p_x, \nu, \lambda_v)$ is true iff $P(p_x, \lambda_v)$ is true. For each state $s \in S$ in M, we define the invariant condition $\hat{Inv}(s) = (c \geq 0)$ in $\hat{M}$. Clearly, $\nu \vDash \hat{Inv}(s)$ for any $\nu \in [C \mapsto \mathbb{Q}]$ and $s \in S$.

Also, any ETTS model can be cast as a TEFSM model. For that, let $M = (S, s_0, X, C, Inv, \longrightarrow)$ be an ETTS model. Take a trivial common domain $\{0\}$ for all parameters and context variables, a single output symbol $Y = \{o\}$ and a single context variable $V = \{v\}$. For each parameter $z \in X \cup Y$, we define $R_z = \{z\}$. Then, the set of z-valuations is the singleton $D_z = \{p_z\}$, where p_z maps z to 0, for all $z \in X \cup \{o\}$. Similarly, $D_V = \{\lambda_v\}$, where λ_v maps v to 0. Now, a transition $t = (s, x, K, \delta, s')$ in M gives rise to a corresponding transition $\hat{t} = (s, x, Q, K, op, o, up, s')$ in $\hat{M}$, where:

- op maps (p_x, λ_v) to p_o;
- up maps (p_x, λ_v) to λ_v; and
- Q maps (p_x, ν, λ_v) to $True$ iff $\nu \vDash \delta$.

Here, the set of invariant conditions Inv for M is the same for $\hat{M}$. The initial state s_0 in $\hat{M}$ is the same initial state s_0 from M.

A configuration of a TEFSM M is a triple (s, ν, λ), where s is a state, ν is a clock interpretation and λ is a context variable valuation. The initial configuration is (s_0, ν_0, λ_0), where s_0 is the initial state of M, ν_0 is the initial clock interpretation of M and λ_0 is an initial context variable valuation of M. A configuration (s, ν, λ) is valid iff $\nu \vDash Inv(s)$. Let $\Gamma \subseteq S \times [C \mapsto \mathbb{Q}] \times D_V$ be the set of configurations of M.

3.2 The Operational Semantics for TEFSM models

Considering the dynamic behavior of ETTS models and the data and control flow of EFSM models, we define the operational semantics of a TEFSM M as follows.

Definition 3.1 Let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2$, be two configurations of M. There is an implicit move from γ_1 to γ_2 iff

(i) $s_1 = s_2$,

(ii) $\lambda_1 = \lambda_2$,

(iii) $\nu_2 = \nu_1 + \tau$, for some $\tau \in \mathbb{Q}^{>0}$, and

(iv) $\nu_2 + \eta \vDash Inv(s_1)$, for all η, $0 \leq \eta \leq \tau$.

We denote such an implicit move by $\gamma_1 \xrightarrow[\tau]{} \gamma_2$.

Definition 3.2 Let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2$, be two configurations of M. Let (x, p_x) be a parameterized input and (y, p_y) be a parameterized output. There is an explicit move from γ_1 to γ_2 over (x, p_x) and yielding (y, p_y) iff there is a transition $(s_1, x, Q, K, op, y, up, s_2)$ in T such that:

(i) $\nu_2 = [K \mapsto 0]\nu_1$,

(ii) $\nu_2 \vDash Inv(s_2)$,

(iii) Q maps (p_x, ν_1, λ_1) to $True$,

(iv) op maps (p_x, λ_1) to p_y, and

(v) up maps (p_x, λ_1) to λ_2.

We denote such an explicit move by $\gamma_1 \xrightarrow{\chi/\xi} \gamma_2$, where $\chi = (x, p_x)$ e $\xi = (y, p_y)$.

Definition 3.3 Let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2, 3$; $\tau \in \mathbb{Q}^{>0}$, (x, p_x) a parameterized input and (y, p_y) a parameterized output. If $\gamma_1 \xrightarrow[\tau]{} \gamma_2$ and $\gamma_2 \xrightarrow{\chi/\xi} \gamma_3$, where $\chi = (x, p_x)$ e $\xi = (y, p_y)$, then we say that there is a move from γ_1 to γ_3 and indicate this by $\gamma_1 \xrightarrow[\tau]{\chi/\xi} \gamma_3$.

Some of the decorations over and under $\longrightarrow$ may be dropped if they are clear from the context.

A parameterized input sequence is any sequence $\bar{\rho} = \rho_1\rho_2\ldots$ where each ρ_i is a parameterized input. A parameterized timed input sequence, or timed input, is a pair $(\bar{\rho}, \bar{\tau})$ where $\bar{\rho}$ is a parameterized input and $\bar{\tau}$ is a time sequence. Similar definitions hold for parameterized outputs. In particular a timed output is a parameterized timed output sequence.

A sequence of configurations $\bar{\gamma} = \gamma_0\gamma_1\gamma_2\ldots$ is a run of M iff there are a timed input $(\bar{\rho}, \bar{\tau})$ and a parameterized output sequence $\bar{\mu}$ such that

$$\gamma_{i-1} \xrightarrow[\theta_i]{\rho_i/\mu_i} \gamma_i, \text{ where } \theta_i = \tau_i - \tau_{i-1}, \text{ for all } i \geq 1.$$

We say that the run is over the timed input $(\bar{\rho}, \bar{\tau})$ and produces the timed output $(\bar{\mu}, \bar{\tau})$. We also say that $(\bar{\mu}, \bar{\tau})$, or $\bar{\mu}$, is produced by M from γ_0 in response to $(\bar{\rho}, \bar{\tau})$.

Some notions from the EFSM and ETTS models are extended to the TEFSM model:

- A TEFSM M is said to be predicate complete if, from any configuration (s, ν, λ) and given any parameterized input (x, p), there is a delay τ and a transition $(s, x, Q, K, op, y, up, s')$ such that Q evaluates $(p, \nu + \tau, \lambda)$ to $True$ and $\nu + \eta \vDash Inv(s)$, for all $0 \leq \eta \leq \tau$.
- The TEFSM M is complete if, for each state s there is a transition leaving s on any input symbol x.
- We say M is deterministic if, for any configuration (s, ν, λ), any parameterized input (x, p), and any time instant τ, there are no two different transitions $(s, x, Q_1, K_1, op_1, y_1, up_1, s_1)$ and $(s, x, Q_2, K_2, op_2, y_2, up_2, s_2)$ such that both Q_1

and Q_2 evaluate $(p, \nu + \tau, \lambda)$ to $True$.

- And, we say M is observable if, for any configuration (p, ν, λ), any parameterized input (x, p) there are no two transitions $(s, x, Q_1, K_1, op_1, y_1, up_1, s_1)$ and $(s, x, Q_2, K_2, op_2, y_2, up_2, s_2)$ with $y_1 \neq y_2$ and with Q_1 and Q_2 both evaluating (p, ν, λ) to $True$.

3.3 Configuration Distinguishability in the TEFSM model

Distinguishability of configurations in the Timed Extended Finite State Machine model is defined over parameterized input sequences. Two configurations γ and γ' of two distinct machines M and M', respectively, are distinguishable over a timed input $(\bar{\rho}, \bar{\tau})$ if the corresponding timed outputs $(\bar{\mu}, \bar{\tau})$ and $(\bar{\mu}', \bar{\tau}')$, produced by M and M' over $(\bar{\rho}, \bar{\tau})$ from γ and γ', respectively, are not compatible, in a sense to be defined shortly. We also say that $(\bar{\rho}, \bar{\tau})$ is a timed input separating those two configurations. We formalize these notions in the sequel, extending the definitions in [27]. Given a context variable valuation λ and a set of variables U, the U-projection of λ is the valuation obtained from λ by retaining the variables that are in the set U, denoted by $\lambda \downarrow U$. Similarly, for input symbols and their valuations, and for output symbols and the corresponding valuations.

Definition 3.4 Let y and y' be outputs of TEFSMs M and M', respectively. Let R and R' be the sets of parameters associated, respectively, with y and y'. The parameterized outputs (y, p) and (y', p') are said to be compatible if $y = y'$ and $p \downarrow R' = p' \downarrow R$. Two parameterized output sequences, $(y_1, p_1) \dots (y_k, p_k)$ of M and $(y'_1, p'_1) \dots (y'_k, p'_k)$ of M' are compatible if, for all $i = 1, \dots, k$, the parameterized outputs (y_i, p_i) and (y'_i, p'_i) are compatible.

Intuitively, parameterized outputs are compatible when the output symbol is the same, and the output valuation agrees on all common output symbols. Distinguishability of configurations is defined as follows.

Definition 3.5 Given a timed input $\bar{\alpha} = (\bar{\rho}, \bar{\tau})$, a configuration γ of M and a configuration γ' of M' are distinguishable by $\bar{\alpha}$ if parameterized output sequence produced by M from γ in response to $\bar{\alpha}$ is not compatible with any parameterized output sequence that can be produced by M' from γ' in response to $\bar{\alpha}$. The timed input $\bar{\alpha}$ is said to be a sequence separating γ from γ'.

4 Timed Extended FSM Product

In Section 5 we extend to TEFSMs the method for the derivation of configuration confirming sequences defined in [27]. Since this method requires the notion of product machines, in this section we present the necessary extension of that notion to TEFSMs.

In the product of TEFSMs, the occurrence of implicit transitions can be ignored, since the global time frame which is used for all clock variables is the same for both

TEFSMs. This guarantees that the system evolution is maintained during implicit transitions.

Let $M^i = (S^i, Inv^i, T^i)$, $i = 1, 2$, and $\gamma^i = (s_0^i, \nu_0^i, \lambda_0^i)$, $i = 1, 2$, be two TEFSMs and their corresponding initial configurations. The product machine is denoted by $M^1 \times M^2$. We will use superscript 1 to denote elements of M^1, like R^1 is the set of parameters for M^1. Likewise, superscript 2 will indicate objects associated with M^2, like V^2 is the set of context variables of M^2. The superscript $1, 2$ is reserved for the product machine $M^1 \times M^2$.

The set of input symbols of $M^{1,2}$ is $X^{1,2} = X^1 \cup X^2$. Likewise, $Y^{1,2} = Y^1 \cup Y^2$. The set of parameters of $M^{1,2}$ is given by $R^{1,2} = R^1 \cup R^2$, with the proviso that for all $z \in R^1 \cap R^2$, the valuations of z in M^1 and M^2 have a common domain. It is clear that we are using the same parameter domains in $M^{1,2}$ as they were in M^1 and M^2. For any $z \in X^{1,2} \cup Y^{1,2}$, we let $R_z^{1,2} = R_z^1 \cup R_z^2$. Note that, given a valuation $r_z^{1,2}$ for elements in $R_z^{1,2}$ we can get valuations $r_z^1 = r_z^{1,2} \downarrow R_z^1$ and $r_z^2 = r_z^{1,2} \downarrow R_z^2$, for machines M^1 and M^2, respectively, and, moreover, $r_z^{1,2} = r_z^1 \cup r_z^2$. Similarly for clock interpretations and context variable valuations. We assume that clocks and context variables are disjoint, i.e., $C^{1,2} = C^1 \cup C^2$, with $C^1 \cap C^2 = \emptyset$, and $V^{1,2} = V^1 \cup V^2$, with $V^1 \cap V^2 = \emptyset$. As for the valuation domains, they are the same as in M^1 as in M^2. The set of states of $M^{1,2}$ is given by $S^{1,2} = S^1 \times (S^2 \cup \{fail\})$, where *fail* is a new state. The set of invariant conditions $Inv^{1,2}$ of $M^{1,2}$ maps $S^{1,2}$ to $\Phi(C^{1,2})$, and it is given by $Inv^{1,2}(s_1, s_2) = Inv^1(s_1) \wedge Inv^2(s_2)$, for all $(s_1, s_2) \in S^{1,2}$. Moreover, $Inv^{1,2}(s_1, fail) = Inv^1(s_1)$, for all $s_1 \in S^1$.

The initial configuration of $M^{1,2}$ will be given by $\gamma_0^{1,2} = ((s_0^1, s_0^2), (\nu_0^{1,2}, \lambda_0^{1,2}))$, where $\nu_0^{1,2} = \nu_0^1 \cup \nu_0^2$ and $\lambda_0^{1,2} = \lambda_0^1 \cup \lambda_0^2$. Note that we can take unions here, since clock and context variables are disjoint in M^1 and M^2.

It remains to specify the transitions of $M^{1,2}$. Let $(s_1^i, x, Q^i, K^i, op^i, y^i, up^i, s_2^i)$, $i = 1, 2$, be transitions of M^1 and M^2, both with the same input x. In the following definition we will be considering a parameterized input $(x, p_x^{1,2})$, a clock interpretation $\nu^{1,2}$ and a context variable valuation $\lambda^{1,2}$, all for the machine $M^{1,2}$. We also let $p_x^1 = p_x^{1,2} \downarrow R_x^1$ and $p_x^2 = p_x^{1,2} \downarrow R_x^2$. Likewise, we let $\nu^1 = \nu^{1,2} \downarrow C^1$ and $\nu^2 = \nu^{1,2} \downarrow C^2$, and also $\lambda^1 = \lambda^{1,2} \downarrow V^1$ and $\lambda^2 = \lambda^{1,2} \downarrow V^2$. There are two cases:

case 1: $y^1 = y^2$ and $op^1(p, \lambda) \downarrow R_{1,2} = op^2(p, \lambda) \downarrow R_{1,2}$, for all $(p, \lambda) \in D_x \times D_V$ where $R_{1,2} = R_{y^1}^1 \cap R_{y^2}^2$. That is, the output symbol is the same and the output valuations of both transitions are the same on each common output parameter. We add two transitions to $T^{1,2}$,

(i) $((s_1^1, s_1^2), x, Q, K, op, y^1, up, (s_2^1, s_2^2))$, where:

(a) $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1) \wedge Q^2(p_x^2, \nu^2, \lambda^2)$

(b) $K = K^1 \cup K^2$

(c) $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1) \cup op^2(p_x^2, \lambda^2)$. Recall that op^1 and op^2 coincide on common output parameters and so we can safely take the union.

(d) $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1) \cup up^2(p_x^2, \lambda^2)$. Recall that $V^1 \cap V^2 = \emptyset$.

(ii) $((s_1^1, s_1^2), x, Q, K, op, y^1, up, (s_2^1, fail))$, where:

(a) $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1) \wedge (\neg Q^2(p_x^2, \nu^2, \lambda^2))$
(b) $K = K^1$
(c) $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1)$
(d) $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1)$

case 2: Else, when the output valuations or the output symbols do not match, we add the transition $((s_1^1, s_1^2), x, Q, K, op, y, up, (s_2^1, fail))$ to $T^{1,2}$, where:
(i) $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1)$
(ii) $K = K^1$
(iii) $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1)$
(iv) $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1)$

Moreover, if $(s_1^1, x, Q, K, op, y, up, s_2^1)$ is a transition of M^1, we add to $M^{1,2}$ the transition $((s_1^1, fail), x, Q, K, op, y, up, (s_2^1, fail))$.

Suppose that the product machine is in the state (s_1^1, s_1^2), and on input $(x, p_x^{1,2})$ we find that M^1, on state s_1^1, has a transition on input (s_1^1, p_x^1), where p_x^1 is the reduction of $p_x^{1,2}$ to the parameters associated with x in M^1. Similarly, M^2, on state s_1^2, has a transition on (x, p_x^2). Moreover, the output of these transitions agree on the output symbol y, and also on valuations of any common output parameter of y in M^1 and in M^2. In this situation, we would want the product machine $M^{1,2}$ to enact both transitions of M^1 and M^2, componentwise. For that: (i) the same clocks are reset; (ii) the output parameter valuations are copied from M^1 and M^2; and (iii) both context updates are also carried over to $M^{1,2}$. But we can only enable this action in $M^{1,2}$ if both transitions in M^1 and M^2 are enabled. This is case 1(i).

Otherwise, we consider the situation where the transition in M^1 is enabled, but the one in M^2 is not. Here, we follow case 1(ii), and make the product machine $M^{1,2}$ enact the behavior of M^1 using for that the first state component, while the second component is marked as *fail*, thereby ignoring the transition from M^2. Note that, in this scenario, M^1 might have taken its transition, while M^2 would be forbidden to do so, even when their external behavior would have been indistinguishable. After the second state component is set to *fail*, $M^{1,2}$ behaves essentially as M^1.

Finally, when the product machine is in state (s_1^1, s_1^2), and we are considering an input $(x, p_x^{1,2})$, and we have picked two transitions from M^1 and M^2, starting respectively at s_1^1 and s_1^2, and whose output symbols or output parameter valuations do not match as above, then we proceed as in case 2. This is similar to case 1(ii) in that the second state component in $M^{1,2}$ is marked as *fail*, and $M^{1,2}$ uses the first state component to behave as M^1, from this moment on.

Consider configurations $\gamma^i = (s^i, \nu^i, \lambda^i)$ of machine M^i, $i = 1, 2$. Let $\bar{\rho} = (\bar{x}, \bar{p_x})$ be a parameterized input sequence for $M^1 \times M^2$, and let $\bar{\alpha} = (\bar{\rho}, \bar{\tau})$ is a timed input for $M^1 \times M^2$. Note that, M^1 and M^2 can be the same machine with different initial configurations. We say that $\bar{\alpha}$ is a separating sequence for γ^1 and γ^2 iff there is a run $\bar{\gamma} = \gamma_0\gamma_1 \ldots$ of $M^{1,2}$ over $\bar{\alpha}$, where $\gamma_0 = ((s^1, s^2), \nu^1 \cup \nu^2, \lambda^1 \cup \lambda^2)$ and for some $i \geq 1$, γ_i is a configuration of $M^{1,2}$ whose state is $(s_j^1, fail)$ for some $s_j^1 \in S^1$.

The problem of determining a separating sequence for two configurations of a

given TEFSM M can be reduced to a reachability problem. The reachability analysis is tractable but hard for EFSMs [23]. Indeed, for TEFSMs it is intractable. This is due to the temporal aspect within the new model. Another difficulty is the combinatorial explosion in the number of states in product machines. Some approaches try to overcome this difficulty by relaxing their restrictions. Approximation algorithms are also used when doing reachability analysis. Other approaches adapted known algorithms in order to manipulate symbolic data structures [34,9,35].

Other simpler contexts [25,3] present algorithms to obtain separating sequences. We postulate that these ideas can be adapted and extended in order to obtain separating sequences in the TEFSM formalism. Such separating sequences would be the result of the test case generation procedure. Moreover, we have been working with the notion of automata discretization in order to overcome the problem of infinite time instants. In addition, it is possible to modify conventional algorithms to reduce the state space generated by the product machine. Another alternative to obtain tractability in a timed approach for finding separating sequences is through the use of suspicious configurations [5]. In this case, we can choose a set of suspicious states, representing a important class fault, based on the expertise of test designers and on assumptions of implementations faults, as seen in [13,31].

5 Test Generation

This section outlines the main concepts for test case generation. First, we present some discussion on the main rationale of conformance testing. Second, we discuss the notion of confirming configurations, and how it is applied. At last, we discuss deriving test sequences by model-checking for TEFSMs.

5.1 Conformance Testing

Conformance testing aims at determining whether an implementation behaves in accordance with a given specification [21,15]. In general, an implementation is regarded as a black box, of which only input/output interfaces are known. In this situation, to verify whether an implementation is in conformance to a specification usually requires an infinite set of test cases in order to exhaust all error possibilities in the implementation. To overcome this problem, one possibility is to define a set of test hypotheses in order to reduce the number of test cases to be considered [13]. Test hypotheses strike a balance between two conflicting aspects. On the one hand, test hypotheses must be defined to be restrictive enough to render the method feasible and tractable. On the other hand, these hypotheses must be as less restrictive as possible, in such a way to be applicable to the largest possible set of implementations.

Conformance testing is guided by a conformance relation between the implementation and the specification [13]. In order to decide whether an implementation is in conformance to a specification, we observe the implementation's outputs to some applied inputs. Considering real-time systems, it must be also verified whether an implementation when stimulated by inputs responds with the expected outputs

within an allowed time interval.

The problem of using a conformance relation is the number of test sequences which should be obtained in order to verify whether each possible implementation is in conformance to a given specification. This problem is worse for timed systems, where there are infinite time instants for a transition to occur. To overcome this problem, we also need to enforce certain hypotheses about the implementation, as discussed in Section 5. This set of hypotheses will reduce the number of possible faults to be considered over the implementation and will render the method feasible in practical cases.

Several methods employ identification sequences to generate test cases from models. An identification sequence has the property of determining the correctness of the configuration reached after some input sequence is taken. Identification sequences may be defined as characterization sets [8,13], as distinguishing sequences [17] or as confirming configuration sequences (CCSs) [27], depending on the model and the generation method. A CCS which are investigated in this paper is a sequence that can increase the confidence that the correct configuration has been reached in the implementation.

5.2 Configuration Confirming Sequences

A configuration confirming sequence (CCS) is a timed input that can be applied to the implementation in order to increase the confidence on its correctness. A CCS can be derived from the product of two machines, one being a specification and the other an undesirable configuration. However, unlike the FSM models where a finite set of undesirable configurations can be postulated, with EFSM models and TEFSM models it is not possible, or desirable, to determine all undesirable configurations. To overcome this problem, a finite set of suspicious configurations is considered [27]. A set of suspicious configurations is derived from the specification to model suspicious implementations which can potentially have faults, reflecting the test designer's assumptions about the implementation faults. The suspicious configurations are extracted from the specification using a set of test hypotheses based on the fault model (e.g., [13]) and relying on the test designer's expertise.

These hypotheses define equivalence classes of implementations that must be put under testing, and they are used to reduce the number of possible implementations that need to be considered. In this work, we assume the following test hypotheses:

(i) Specifications and suspicious implementations are modeled by TEFSMs;

(ii) The number of clocks in the specification must be less than or equal to the number of clocks in the suspicious implementations; and

(iii) The same alphabets are used in both specification and suspicious implementations.

Given a configuration and a suspicious configuration, deriving a CCS can be reduced to the problem of finding a path in the product of two distinct TEFSMs, or of the same core TEFSM with distinct initial configurations. Such a sought

path would run from the initial state to a fail state. If the fail state can not be reached, then the suspicious configuration is equivalent to the original configuration. However, if a fail state is reachable, the model-checking algorithm will produce a counter-example, as a sequence of transitions that leads to this fail state [10]. This sequence would make a test case for the suspicious configuration. However, it is still necessary to identify in which moment each transition was taken, as well as the valuation of the input parameters associated with each input symbol. Gathering of this information forms a set of test cases. The test case is then used to exercise a real implementation, and the outputs are compared with the outputs produced by the specification over the same data. If a disagreement is found between corresponding outputs, then a fault has been identified.

5.3 Model-checking

Design errors frequently occur when conventional simulation and testing techniques are used to check safety systems. Model checking is a set of techniques for the automatic analysis of reactive and real-time systems. Some model checking algorithms have been modified to discover such errors, thus providing more quality and accuracy in system verifications. In general, given a transition system and a property, the model checking problem is to decide whether the property holds or not in the model represented by the transition system. If not, the model checking algorithm provides a counterexample, i.e. an execution over the model that violates the property [23].

Reachability analysis is a special kind of model-checking method that can be applied in a formal model. In general, given a special state to be found in a model, the reachability analysis decides if it is possible to move from the initial state to the final special state.

To summarize, to automatically test implementations based on a specification represented by a machine M, the following steps are performed:

(i) An empty set TC of test cases is defined.

(ii) Given a configuration γ of M, a set of suspicious configurations Γ is defined, based on test hypotheses, fault models and some specific test engineer's objectives.

(iii) For each suspicious configuration $\gamma_s \in \Gamma$, the product of M with itself is constructed, having γ as the initial configuration of the first instance of M in the product, and having γ_s as the initial configuration of the second instance.

(iv) Reachability analysis is carried out, in order to find a path to a fail state in the product machine. If such a path is found, it is added to TC.

(v) For each $tc \in TC$, a time and an input parameter valuation sequences are derived so as to satisfy the predicates along the path specified by tc.

(vi) Each path in TC, with its associated data, is applied to the real implementation under testing.

6 An Example

We are given two TEFSMs M and N, where M is a specification and N is a suspicious implementation of M. We obtain the product of these machines, $M \times N$, by applying our method. In this example, as is usual in practice, N has the same transitions as M. They differ only in their associated initial configurations. Accordingly, we will denote the product by $M^0 \times M^1$, where M^0 is the specification and M^1 is the suspicious configuration. The TEFSM M depicted in Figure 1.

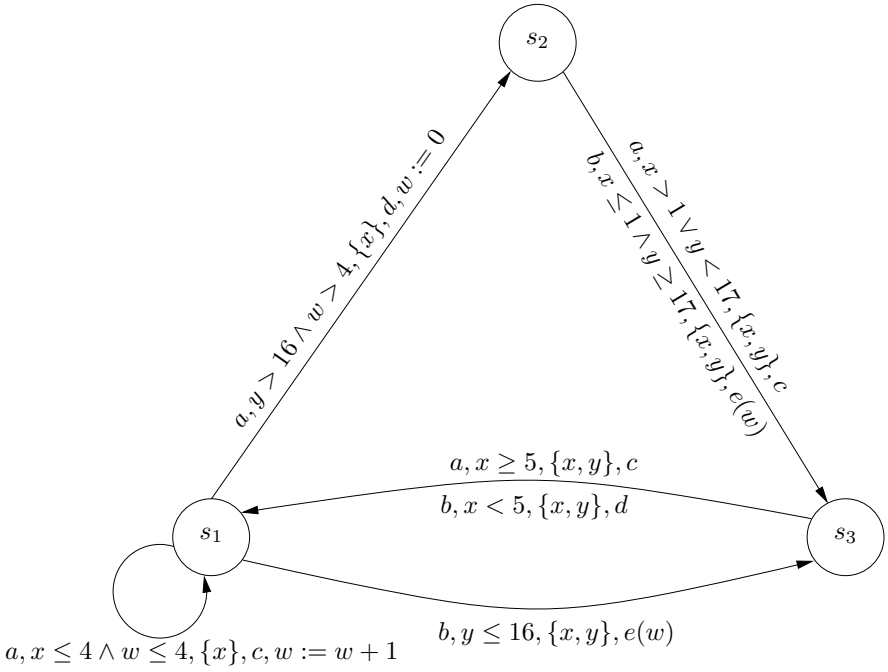

Fig. 1. The TEFSM M.

It has three states and seven transitions. The input set is $\{a, b\}$ and the output set is $\{c, d, e\}$. Furthermore, M has two clock variables, x and y, and one context variable w. There are no parameters associated with the input symbols, i.e. $R_a = R_b = \emptyset$. Likewise, $R_c = R_d = \emptyset$. For each state s in M, the control remains in s whenever its invariant condition is satisfied. The output e has only one associated parameter. In this case, it is not necessary to name the parameter. Instead, in Figure 1 and in the sequel we write $e(w)$ to indicate that the current value of the context variable w is to be attributed to the parameter associated with e. In the figure, each arrow is labeled by a sequence of items. The first three are always the input symbol, the predicate function and the set of clocks to be reset in the transition, respectively. Next, comes the output symbols, either c or d, and we write directly $e(w)$ to indicate both the output symbol and the value of its parameter. Finally, if the value of the context variable w is altered by the transition, this is indicated by the attribution that appears at the end of the label; if the value of w is not altered by the transition we simply omit the trivial expression $w := w$.

A configuration of M is given by a state, a clock interpretation and a context

variable valuation. Hence, a configuration of M will be denoted by $(s,(n,m),k)$ indicating that the machine is in state s, n and m are the values for the clock variables x and y, respectively, and k is the value for the context variable w. The integers are selected as a common valuation domain. In the configuration $(s_1,(3,2),4)$ the transition $a, x \leq 4 \wedge w \leq 4, \{x\}, c, w := w+1$, from s_1 to itself, is enabled. Likewise, the transition $b, y \leq 16, \{x,y\}, e(w)$, from s_1 to s_3, is also enabled.

For the product, let M^0 designate M with the initial configuration $(s_1,(0,0),2)$, and let M^1 designate M with initial configuration $(s_1,(4,2),5)$. The TEFSM product of $M^0 \times M^1$ is shown in Figure 2. To simplify the notation in the example, we will use subscript i to denote items of machine M^i, for $i = 0, 1$, e.g. x_1 represents the clock variable x of M_1, while w_0 denotes the variable w in M^0.

The initial configuration of $M^0 \times M^1$ is denoted by $((s_1,s_1),(0,0,2,4),(2,5))$, where we list first the items corresponding to M^0, followed by the items associated with M^1. Note that, in the figure, states are represented by subscripts, e.g., the state (s_1,s_1) in the product is named s_{11}.

By inspection of the product, we can see that the input b enables the transition to fire, since clock conditions on y_0 and y_1 are satisfied for the initial configuration. After that the transition is taken, the new configuration is given by $((s_3,s_3),(0,0,2,0),(0,5))$. It is easy to see that neither input a nor input b will enable transitions to fire so as to reach, directly, a fail state. Note that, every clock variable was reset to zero, and transition guards are excluding for clock variables x_0 and x_1. If the input b occurs within less than 5 time units, the configuration becomes $((s_1,s_1),(0,0,2,0),(0,5))$. Otherwise, if the time evolves for more than 5 time units, only the input a could stimulate the machine to change configurations. The new configuration would still be $((s_1,s_1),(0,0,2,0),(0,5))$. Both transitions would drive the control back to the initial state, where the transition stimulated by input b is the unique one enabled to fire. This cycle would be executed repeatedly and a fail state would not be reached.

Another possibility is to take the transition on the input a. It is easy to see that the input a separates the configurations $(0,0,2)$ from $(4,2,5)$. The final configuration reached is $((s_1,fail),(0,0,3,4),(2,5))$. On the other hand, the control can be kept within the state (s_1,s_1), by a continuous time evolution. After that, the stimulation by input a enables the transition to fire, and the configuration $((s_1,s_1),(1,1,2,5),(3,5))$ can be reached. Then, the transition from state (s_1,s_1), on input a and with associated predicate $x_0 \leq 4 \wedge w_0 \leq 4 \wedge (w_1 > 4 \vee x_1 > 4)$ is enabled and takes the machine to the fail state $(s_1,fail)$. Here, only the clock variable x_0 is reset, and the context variable w_0 is updated by one unit. The new configuration will be $((s_1,fail),(0,1,3,5),(3,5))$. From here, we see that input a separates the configuration $(0,0,2)$ from $(4,2,5)$, after some time passes. The new configuration that can be reached in this case is $((s_1,fail),(0,1,3,5),(3,5))$.

If we consider another situation, where the initial configurations of M^0 and M^1, respectively, are given by $(0,0,2)$ and $(4,2,4)$, another run of $M^0 \times M^1$ will also reach the fail state. In this case, a reachability analysis shows that the fail state of $M^0 \times M^1$ can only be reached when a sequence of one or two consecutive inputs a

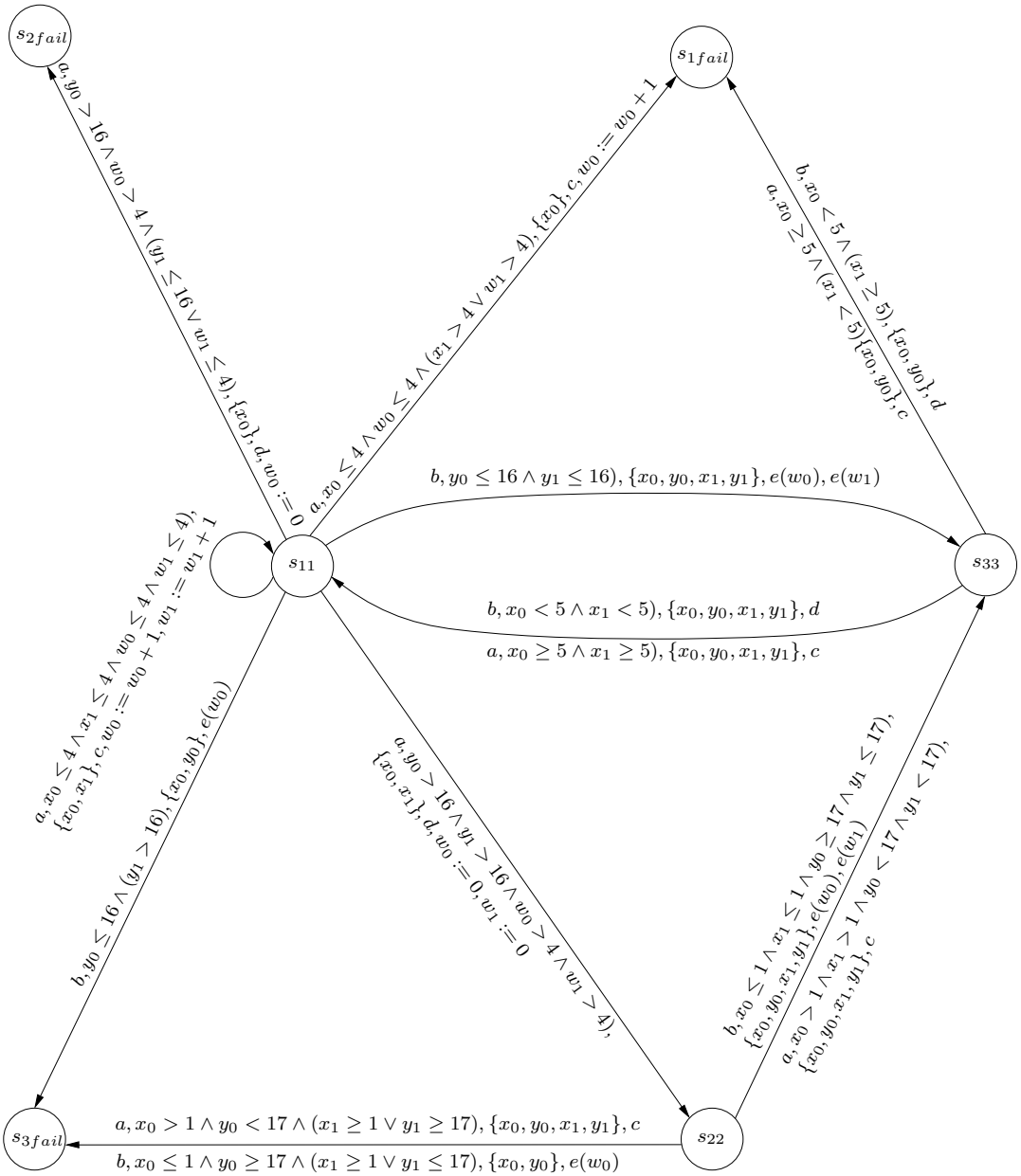

Fig. 2. The TEFSM product of M with itself.

is applied.

In the example, M^0 represents the specification, M^1 represents a suspicious implementation, and the product $M^0 \times M^1$ is used to find sequences of configurations that show non conformance between a suspicious implementation and the specification. We can derive traces from the reachability analysis of $M^0 \times M^1$. The resulting traces are runs that reach the fail state in the product machine, starting from the initial configurations of the participating TEFSMs.

7 Concluding Remarks

The ability to derive test cases from formal models opens the possibility that we can construct more rigorous and dependable systems, by providing a sound basis for the validation of the systems' behaviors. There is a direct relationship between the kinds of systems that a given model can deal with and the availability of methods for deriving test cases. The FSM and EFSM models are well-established and have been intensively investigated. One important feature they both lack is the ability to deal with time. In this paper we define TEFSMs as a model that extends the EFSM model with the notion of time. From that, we discussed an extended method for deriving configuration confirming sequences for TEFSMs, a step toward automating the generation of test cases from these models.

Although we can argue that both the model and the generation method can be used, we do not have answers for pragmatic questions, such as (i) how difficult is it to describe a system using TEFSMs and (ii) how large are the models we can handle. To answer these questions, it is necessary to deepen the investigations and implement adequate supporting software tools. We are currently working in this direction.

Other aspects that can be investigated include how to allow time constraint to be defined over outputs. We note that our definition does not deal with constraints that may reflect output response that is not instantaneous. The input and output occur in the same time instant. We are considering how this extension might impact the test case generation methods.

References

[1] Alur, R., *Timed automata*, in: *CAV*, number 1633 in LNCS, 1999, pp. 8–22.

[2] Alur, R. and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.
URL citeseer.ist.psu.edu/alur94theory.html

[3] Alur, R., M. McDougall and Z. Yang, *Exploiting behavioral hierarchy for efficient model checking.*, in: *CAV*, 2002, pp. 338–342.

[4] Behrmann, G., K. G. Larsen, J. Pearson, C. Weise and W. Yi, *Efficient timed reachability analysis using clock difference diagrams*, in: *Computer Aided Verification*, 1999, pp. 341–353.
URL citeseer.ist.psu.edu/article/behrmann99efficient.html

[5] Bonifácio, A. L., A. V. Moura, A. d. S. Simão and J. C. Maldonado, *Conformance Testing by Model Checking Timed Extended Finite State Machines*, in: *Brazilian Symposium on Formal Methods (SBMF'06)*, Natal, 2006, pp. 43–58.

[6] Campos, S. V., M. Minea, W. Marrero, E. M. Clarke and H. Hiraishi, *Computing quantitative characteristics of finite-state real-time systems*, in: *Proc. 15th IEEE Real-Time Systems Symp.* (1994), pp. 266–270, san Juan, Porto Rico.

[7] Cardell-Oliver, R., *Conformance tests for real-time systems with timed automata specifications*, Formal Aspects of Computing **12** (2000), pp. 350–371.
URL citeseer.ist.psu.edu/385816.html

[8] Chow, T. S., *Testing software design modeled by finite-state machines*, IEEE Transactions on Software Engineering **4** (1978), pp. 178–187.

[9] Cimatti, A., E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, *Integrating bdd-based and sat-based symbolic model checking.*, in: *FroCos*, 2002, pp. 49–56.

[10] da Silva, D. A. and P. D. L. Machado, *Towards test purpose generation from ctl properties for reactive systems.*, Electr. Notes Theor. Comput. Sci. **164** (2006), pp. 29–40.

[11] Dickhöfer, M. and T. Wilke, *Timed alternating tree automata: the automata-theoretic solution to the tctl model checking problem*, in: *26th ICALP*, LNCS **1644**, 1999, pp. 281–290.
URL citeseer.ist.psu.edu/article/dickhfer99timed.html

[12] Dorofeeva, R., K. El-Fakih and N. Yevtushenko, *An improved conformance testing method*, in: *Formal Techniques for Networked and Distributed Systems*, Lecture Notes in Computer Science **3731** (2005), pp. 204–218.

[13] En-Nouaary, A., R. Dssouli and F. Khendek, *Timed wp-method: Testing real-time systems*, IEEE Trans. Softw. Eng. **28** (2002), pp. 1023–1038.

[14] Fujiwara, S., G. V. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi, *Test selection based on finite state models*, IEEE Transaction on Software Engineering **17** (1991).

[15] Gargantini, A., *Conformance testing*, in: M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, Lecture Notes in Computer Science **3472** (2005), pp. 87–111.

[16] Gill, A., "Introduction to the theory of finite-state machines," McGraw-Hill, New York, 1962.

[17] Gonnenc, G., *A method for the design of fault detection experiments*, IEEE Transactions on Computing **19** (1970), pp. 551–558.

[18] Higashino, T., A. Nakata, K. Taniguchi and A. R. Cavalli, *Generating test cases for a timed i/o automaton model*, in: *Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems* (1999), pp. 197–214.

[19] Hirai, T., *An application of temporal linear logic to Timed Petri Nets*, in: *Proceedings of the Petri Nets'99 Workshop on Applications of Petri Nets to Intelligent System Development*, 1999, pp. 2–13.

[20] Krichen, M., *State identification*, in: M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, Lecture Notes in Computer Science **3472** (2005), pp. 87–111.

[21] Krichen, M. and S. Tripakis, *Black-box conformance testing for real-time systems*, in: *Model Checking Software: 11th International SPIN Workshop*, number 2989 in Lecture Notes in Computer Science, Barcelona, Spain, 2004, pp. 109–126.

[22] McMillan, K. L., "Symbolic Model Checking: An Approach to the State Explosion Problem," Kluwer Academic, 1993.

[23] Merz, S., *Model checking: A tutorial overview*, in: F. C. et al., editor, *Modeling and Verification of Parallel Processes*, Lecture Notes in Computer Science **2067**, Springer-Verlag, Berlin, 2001 pp. 3–38.

[24] Møller, J. B., *Simplifying fixpoint computations in verification of real-time systems* (2002).
URL http://citeseer.ist.psu.edu/540135.html

[25] Nr, B., M. Dickhofer and T. Wilke, *The automata-theoretic method works for TCTL model checking* (1998).
URL http://citeseer.ist.psu.edu/44733.html

[26] Offutt, A. J., Y. Xiong and S. Liu, *Criteria for generating specification-based tests*, in: *Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '99)*, Las Vegas, NV, 1999, pp. 41–50.

[27] Petrenko, A., S. Boroday and R. Groz, *Confirming configurations in efsm testing*, IEEE Trans. Softw. Eng. **30** (2004), pp. 29–42.

[28] Petrenko, A. and N. Yevtushenko, *Testing from partial deterministic fsm specifications*, IEEE Transactions on Computers **54** (2005).

[29] Tretmans, J., *Test generation with inputs, outputs, and quiescence.*, in: T. Margaria and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, Lecture Notes in Computer Science **1055** (1996), pp. 127–146.

[30] Tretmans, J., *Testing concurrent systems: A formal approach*, in: J. Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, Lecture Notes in Computer Science **1664** (1999), pp. 46–65.

[31] Wang, C.-J. and M. T. Liu, *Generating test cases for efsm with given fault models.*, in: *INFOCOM*, 1993, pp. 774–781.

[32] Wang, F., *Efficient verification of timed automata with bdd-like data structures.*, STTT **6** (2004), pp. 77–97.

[33] Wang, F., *Formal verification of timed systems: A survey and perspective.*, Proceedings of the IEEE **92** (2004), pp. 1283–1307.

[34] Wang, F., *Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures*, Software Engineering, IEEE Transactions on **31** (2005), pp. 38–51.

[35] Wang, F., *Under-approximation of the Greatest Fixpoints in Real-Time System Verification*, ArXiv Computer Science e-prints (2005), pp. 1060–+.

[36] Wang, F., G.-D. Hwang and F. Yu, *Tctl inevitability analysis of dense-time systems.*, in: *CIAA*, 2003, pp. 176–187.