Conformance Testing Relations for Timed Systems*

Manuel Núñez and Ismael Rodríguez

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
e-mail: {mn,isrodrig}@sip.ucm.es

Abstract. This paper presents a formal framework to test both the functional and temporal behaviors in systems where temporal aspects are critical. Different implementation relations, depending on both the interpretation of time and on the (non-)determinism of specifications and/or implementations, are presented and related. We also study how tests cases are defined and applied to implementations. A test derivation algorithm, producing sound and complete test suites, is presented.

1 Introduction

The complexity of current systems necessarily leads to a higher relevance of testing issues during the development project. The scale and heterogeneity of present projects makes it impossible for developers to have an overall view of the system. Thus, it is difficult to foresee those errors that are either critical or more probable. Since the construction of a system requires to use several components, developed by different teams, reliability of these components is a must. This is a requirement not only for final customers but also for developers. In this context, *formal testing techniques* provide systematic procedures to check implementations in such a way that the coverage of critical parts/aspects of the system depends less on the intuition of the tester.

The application of formal testing techniques to check the correctness of a system requires to identify the *critical* aspects of the system, that is, those aspects that will make the difference between correct and incorrect behavior. While the relevant aspects of some systems only concern *what* they do, in some other systems it is equally relevant *how* they do what they do. For instance, the probability of an event to happen may be considered critical in a non-deterministic system. If a system runs in an environment where computational resources are shared by several systems, the consumption of resources may be relevant as well. Similarly, the time consumed by each operation should be considered critical in

* Research partially supported by the Spanish *Ministerio de Ciencia y Tecnología* project MASTER (TIC2003-07848-C02-01), the *Junta de Comunidades de Castilla-La Mancha* project DISMEF (PAC-03-001), and the Marie Curie Research and Training Network *TAROT* (MRTN-CT-2003-505121).

a real-time system. Actually, some operations that are concluded after a given deadline could be useless or unacceptable for the user of the system.

In this paper we present a formal testing methodology where the temporal behavior of systems is considered. A simple extension of the classical concept of *Extended Finite State Machine* will allow a specifier to explicitly denote temporal requirements for each action of a system. We study *conformance testing* relations to relate implementations, belonging to a given set `Imp`, with specifications, taken from another set `Spec`. Our study considers time extensions of the relation $\texttt{conf}_{nt}$ [NR02], which is based on the update of `conf` [BSS86] to deal with inputs and outputs: `ioco` [Tre96,Tre99]. In order to cope with time, we do not take into account only that a system may perform a given action but we also record the amount of time that the system needs to do so. Unfortunately, conformance testing relations for timed systems have not been yet extensively studied, and only very recently some work has been performed in this line (e.g. [NR02,BB04,LMN04]). We propose five timed conformance relations according to the interpretation of *good* implementation for a given specification. Regarding our relations, time aspects add some extra complexity. For example, even though an implementation I had the same traces as a specification S, we should not consider that I conforms to S if the implementation is always *slower* than the specification. Moreover, it can be the case that a system performs the same sequence of actions for different times. These facts motivate the definition of several conformance relations. For example, it can be said that an implementation conforms to a specification if the implementation is always *faster*, or if the implementation is at least as *fast* as the worst case of the specification. We think that the relations that we introduce in this paper can be useful for the study of conformance for other models of timed systems. For example, the definitions can be easily adapted to timed automata [AD94]. Other definitions of timed I/O automata (e.g. [HNTC99,SVD01]) are restricted to deterministic (regarding actions) behavior. In this case, some of our relations will be equivalent among them (i.e. they will relate the same automata).

Regarding the application of testing to timed systems, several proposals have appeared in the literature (e.g. [MMM95,CL97,HNTC99,SVD01,EDK02,ED03]). Our proposal differs from these ones in several points, mainly because the treatment of time is different. We do not have a notion of clock(s) together with time constraints; we associate time to the execution of actions (representing the time that it takes for a system to perform an action). Besides, the time that a transition needs to be performed is not fixed (e.g. the transition t takes 3 units of time). This time depends on the values of the variables. In fact, since those values may change after each transition, it may happen that if we perform two (or more) times a transition then each performance takes a different amount of time. With respect to the application of tests to implementations, the above mentioned non-deterministic temporal behavior of specifications and/or implementations requires that tests work in a specific manner. For example, if we apply a test and we observe that the implementation takes less time than the one required by the specification, then this single application of the test allows

us to know that the implementation *may* be faster than the specification, but not that it *must* be so.

The rest of the paper is organized as follows. In Section 2 we present our model to represent timed systems. In Section 3 we study implementation relations for our framework where temporal behavior is taken into account. We also relate these implementation relations. In Section 4 we show how test cases are defined and describe how to apply them to implementations. In Section 5 we introduce a test derivation algorithm to produce sound and complete, with respect to three of our conformance relations, test suites. Next, in Section 6, we consider two additional relations that can be used when implementations show non-deterministic behavior. We also relate these new relations with the previous ones. Finally, in Section 7 we present our conclusions and some directions for further research.

2 A timed extension of the EFSM model

In this section we introduce our timed extension of the classical extended finite state machine model. The main difference with respect to usual EFSMs consists in the addition of *time*. In order to represent our timed EFSMs we consider that the number of different variables is equal to m. We will assume that each variable x_i belongs to the domain D_i. Thus, the values of all the variables at a given point of time can be represented by a tuple belonging to the cartesian product $D_1 \times D_2 \times \cdots \times D_m$. Regarding the domain to represent time, we consider that time values belong to a certain domain Time (e.g. we may take a continuous domain such as $\mathbb{R}_+$ or a discrete domain such as $\mathbb{N}$).

Definition 1. Let Time be the domain to define time values, $D_1, \ldots, D_m$ be sets of values, and let us consider $\mathcal{D} = D_1 \times D_2 \times \cdots \times D_m$. A *Timed Extended Finite State Machine*, in the following TEFSM, is a tuple $M = (S, I, O, Tr, s_{in}, \bar{y})$ where S is a finite set of states, I is the set of input actions, O is the set of output actions, Tr is the set of transitions, s_{in} is the initial state, and $\bar{y} \in \mathcal{D}$ is a tuple of variables.

Each transition $t \in Tr$ is a tuple $t = (s, s', i, o, Q, Z, C)$ where $s, s' \in S$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, respectively, associated with the transition, $Q : \mathcal{D} \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, $Z : \mathcal{D} \longrightarrow \mathcal{D}$ is a transformation over the current variables, and $C : \mathcal{D} \longrightarrow \texttt{Time}$ is the time that the transition needs to be completed.

A *configuration* in M is a pair $(s, \bar{x})$ where $s \in S$ is the current state and $\bar{x} \in \mathcal{D}$ is the tuple containing the current value of the variables.

We say that $tr = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ is a (timed) *trace* of M if there exist transitions $t_1, \ldots, t_r \in Tr$ such that $t_1 = (s, s_1, i_1, o_1, Q_1, Z_1, C_1)$,..., $t_r = (s_{r-1}, s', i_r, o_r, Q_r, Z_r, C_r)$, the predicate Q is defined such that it holds $Q(\bar{x}) = (Q_1(\bar{x}) \wedge Q_2(Z_1(\bar{x})) \wedge \ldots \wedge Q_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots)))$, the transformation Z is defined as $Z(\bar{x}) = Z_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$, and C is defined as $C(\bar{x}) = C_1(\bar{x}) + C_2(Z_1(\bar{x})) + \cdots + C_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$.

We say that $i_1/o_1, \ldots, i_r/o_r$ is a *non-timed evolution*, or simply *evolution*, of M if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M such that $Q(\bar{y})$ holds. We denote by $\texttt{NTEvol}(M)$ the set of non-timed evolutions of M. We say that the pair $((i_1/o_1, \ldots, i_r/o_r), v)$ is a *timed evolution* of M if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M such that $Q(\bar{y})$ holds and $v = C(\bar{y})$. We denote by $\texttt{TEvol}(M)$ the set of timed evolutions of M. Let $e \in \texttt{NTEvol}(M)$ and $v \in \texttt{Time}$ be such that $(e, v) \in \texttt{TEvol}(M)$. We say that (e, v) is an *instance* of e.

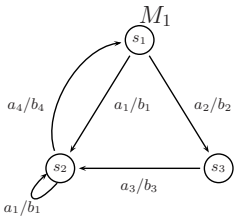
We say that M presents *non-observable non-deterministic* behavior if there exist $s, s_1, s_2 \in S$, $i \in I$, $o \in O$, $Q_1, Q_2 : \mathcal{D} \longrightarrow \texttt{Bool}$, $Z_1, Z_2 : \mathcal{D} \longrightarrow \mathcal{D}$, and $C_1, C_2 : \mathcal{D} \longrightarrow \texttt{Time}$ such that $(s, s_1, i, o, Q_1, Z_1, C_1), (s, s_2, i, o, Q_2, Z_2, C_2) \in Tr$. □

Intuitively, for a configuration $(s, \bar{x})$, a transition $t = (s, s', i, o, Q, Z, C)$ indicates that if the machine is in the state s, receives the input i, and the predicate Q holds for $\bar{x}$, then after $C(\bar{x})$ units of time the machine emits the output o and the values of the variables are transformed according to Z. Timed traces are defined as sequences of transitions. In this case, the predicate, the transformation function, and the time associated with the trace are computed from the ones corresponding to each transition belonging to the sequence. Let us note that different instances of the same evolution may appear in a specification as result of the different configurations produced after traversing the corresponding TEFSM. Finally, let us remark that the notion of non-observable non-determinism is less restrictive than the notion of observable non-determinism. For example, it allows to have both the transitions $(s, s_1, i, o_1, Q_1, Z_1, C_1)$ and $(s, s_2, i, o_2, Q_2, Z_2, C_2)$, as long as $o_1 \neq o_2$.

Example 1. In Figure 1 we present two TEFSMs. For example, let us suppose that the initial value of variables is $\bar{x} = (2, 0, 0, 2)$ and that the initial state of M_1 is s_1. Then, the transition t_{12} can be performed and it will take time $\frac{1}{2}$. After that, the value of the variables will be given by the tuple $(3, 0, 0, 1)$. □

3 (Timed) Implementation Relations

In this section we introduce our implementation relations. These relations are appropriate timed extensions of the relation $\texttt{conf}_{nt}$ [NR02]. All of them follow the same pattern: An implementation I *conforms* to a specification S if for any possible evolution of S the outputs that the implementation I may perform after a given input are a subset of those for the specification. This pattern is borrowed from ioco [Tre96,Tre99] but we do not consider *quiescent* states (that is, states where no external outputs are available). In addition to the non-timed conformance of the implementation, we require some time conditions to hold (this is a major difference with respect to ioco where time is not considered). For example, we may ask an implementation to be always faster than the time constraints imposed by the specification. The different considerations of time produce that there is not a unique way to define an implementation relation.



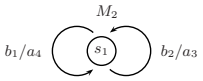
$I = \{a_1, a_2, a_3, a_4\}, O = \{b_1, b_2, b_3, b_4\}$

$t_{12} = (s_1, s_2, a_1, b_1, Q_1, Z_1, C_1)$
$t_{13} = (s_1, s_3, a_2, b_2, Q_2, Z_2, C_2)$
$t_{32} = (s_2, s_3, a_3, b_3, Q_3, Z_3, C_3)$
$t_{21} = (s_2, s_1, a_4, b_4, Q_4, Z_4, C_4)$
$t_{22} = (s_2, s_2, a_1, b_1, Q_5, Z_5, C_5)$

$$Z_i(\bar{x}) = \bar{x} + \begin{cases} (1,0,0,-1) \text{ if } i \in \{1,2,5\} \\ (0,1,-1,0) \text{ if } i \in \{3,4\} \end{cases}$$

$$Q_i(\bar{x}) \equiv Z_i(\bar{x}) \geq \bar{0} \wedge \begin{cases} x_i > 0 \text{ if } i \in \{1,2,3,4\} \\ x_1 > 0 \text{ if } i = 5 \end{cases}$$

$$C_i(\bar{x}) = \begin{cases} \frac{1}{x_i} \text{ if } i \in \{1,2,3,4\} \wedge x_i \neq 0 \\ \frac{1}{x_1} \text{ if } i = 5 \wedge x_1 \neq 0 \end{cases}$$



$I = \{b_1, b_2\}, O = \{a_3, a_4\}$

$t_1 = (s_1, s_1, b_1, a_4, Q_1, Z_1, C_1)$
$t_2 = (s_1, s_1, b_2, a_3, Q_2, Z_2, C_2)$

$$Z_i(\bar{x}) = \bar{x} + \begin{cases} (1,1,-1,-1) \text{ if } i = 1 \\ (1,0,0,0) \text{ if } i = 2 \end{cases}$$

$$Q_i(\bar{x}) \equiv Z_i(\bar{x}) \geq \bar{0} \wedge \begin{cases} x_1 > 0 \wedge x_2 > 0 \text{ if } i = 1 \\ x_3 > 0 \text{ if } i = 2 \end{cases}$$

$$C_i(\bar{x}) = \begin{cases} \frac{1}{x_i \cdot x_2} \text{ if } i = 1 \wedge x_1 \neq 0 \wedge x_2 \neq 0 \\ \frac{1}{x_3} \text{ if } i = 2 \wedge x_3 \neq 0 \end{cases}$$

We suppose that $\bar{x} \in \mathbb{R}^4_+$. We denote by x_i the *i-th* component of $\bar{x}$.

Fig. 1. Examples of TEFSM.

Next, we formally define the sets of specifications and implementations: Spec and Imp. A specification is a timed extended finite state machine. Regarding implementations, we consider that they are also given by means of TEFSMs. In this case, we assume, as usual, that all the input actions are always enabled in any state of the implementation. Thus, we can assume that for any input i and any state of the implementation s there always exists a transition $(s, s, i, \texttt{null}, Q, Z, C)$ where null is a special (empty) output symbol, $Q(\bar{x}) \equiv \neg \bigvee \{Q'(\bar{x}) | \exists$ a transition $(s, s', i, o, Q', Z', C')\}$, $Z(\bar{x}) = \bar{x}$, and $C(\bar{x}) = 0$. Let us note that such a transition will be performed when (and only) no other transition is available (that is, either there are no transitions outgoing from s or none of the corresponding predicates hold). Other solutions consist in adding a transition leading to an *error* state or generating a transition to the initial state. In addition, we will initially consider that implementations may not present non-observable non-deterministic behavior (see Definition 1). The removal of this restriction gives raise to the definition of two additional conformance relations.

First, we recall the implementation relation $\texttt{conf}_{nt}$ where time is not considered.

Definition 2. Let S and I be two TEFSMs. We say that I *non-timely conforms* to S, denoted by $I\ \texttt{conf}_{nt}\ S$, if for each non-timed evolution $e \in \texttt{NTEvol}(S)$ with $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$, $r \geq 1$, we have that

$$e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r) \in \texttt{NTEvol}(I) \text{ implies } e' \in \texttt{NTEvol}(S)$$

□

Next, we introduce our timed implementation relations. In the $\texttt{conf}_a$ relation (conforms *always*) we consider that for any timed evolution of the implementation (e, t) we have that if e is a non-timed evolution of the specification then (e, t) is also a timed evolution of the specification. In the $\texttt{conf}_w$ relation (conforms in the *worst* case) the implementation is forced, for each timed evolution fulfilling the previous conditions, to be faster than the slowest instance of the same evolution in the specification. The $\texttt{conf}_b$ relation (conforms in the *best* case) is similar but considering the fastest instance.

Definition 3. Let S and I be two a TEFSMs. We define the following implementation relations:

- $I\ \texttt{conf}_a\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all timed evolution $(e, t) \in \texttt{TEvol}(I)$ we have $e \in \texttt{NTEvol}(S) \Longrightarrow (e, t) \in \texttt{TEvol}(S)$.
- $I\ \texttt{conf}_w\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all timed evolution $(e, t) \in \texttt{TEvol}(I)$ we have $e \in \texttt{NTEvol}(S) \Longrightarrow (\exists\ t' : (e, t') \in \texttt{TEvol}(S)\ \wedge\ t \leq t')$.
- $I\ \texttt{conf}_b\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all timed evolution $(e, t) \in \texttt{TEvol}(I)$ we have $e \in \texttt{NTEvol}(S) \Longrightarrow (\forall\ t' : ((e, t') \in \texttt{TEvol}(S) \Longrightarrow t \leq t'))$.

□

Theorem 1. [NR02] The relations given in Definition 3 are related as follows:

$$I\ \texttt{conf}_a\ S \Rightarrow I\ \texttt{conf}_w\ S \Leftarrow I\ \texttt{conf}_b\ S$$

□

It is interesting to note that if specifications are restricted to take always the same time for each given evolution (independently from the possible derivation taken for such evolution) then the relations $\texttt{conf}_b$ and $\texttt{conf}_w$ would coincide, but they would be still different from the $\texttt{conf}_a$ relation.

Lemma 1. Let $M = (S, I, O, Tr, s_{in}, \bar{y})$ be a TEFSM. Let us suppose that there do not exist $((i_1/o_1, \ldots, i_r/o_r), t), ((i_1/o_1, \ldots, i_r/o_r), t') \in \texttt{TEvol}(M)$ with $t \neq t'$. For any a TEFSM I we have $I\ \texttt{conf}_b\ M$ iff $I\ \texttt{conf}_w\ M$. □

4 Definition and Application of Test Cases

A test represents a sequence of inputs applied to an implementation under test. Once an output is received, we check whether it belongs to the set of expected ones or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and output sets I and O, respectively, tests are deterministic acyclic I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In the first case we add a *time stamp*. This time will be contrasted with the one that the implementation took to arrive to that point.

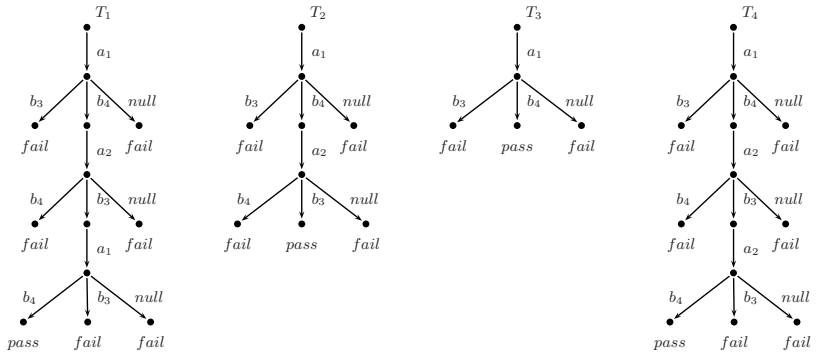


Fig. 2. Examples of Test Cases.

Definition 4. A *test case* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times I \cup O \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For all input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.
- S_O is the set of *output* states. For all output state $s \in S_O$ we have that for all $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I, s' \in S$ such that $(s, i, s') \in Tr$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. Besides, for all state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, $C : S_P \longrightarrow \texttt{Time}$ is a function associating time stamps with passing states.

Let $\sigma = i_1/o_1, \ldots, i_r/o_r$. We write $T \stackrel{\sigma}{\Longrightarrow} s$, if $s \in S_F \cup S_P$ and there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s)\} \subseteq Tr$, for all $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for all $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

We say that a test case T is an *instance* of the test case T' if they only differ in the associated function C assigning times to passing states.

We say that a test case T is *valid* if the graph induced by T is a tree with root at the initial state s_0. □

In Figure 2 we present some examples of test cases (time stamps are omitted). Next we define the application of a tests suite (i.e. a set of tests) to an implementation. We say that the tests suite $\mathcal{T}$ is *passed* if for all test the terminal states reached by the composition of implementation and test are *pass* states. Besides, we give timing conditions according to the different implementation relations.

Definition 5. Let I be a TEFSM, T be a valid test, and s^T be a state of T. We write $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ if $T \stackrel{\sigma}{\Longrightarrow} s^T$ and $(\sigma, t) \in \mathtt{TEvol}(I)$.

We say that I *passes* the set of tests $\mathcal{T}$, denoted by $\mathtt{pass}(I, \mathcal{T})$, if for all test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ and $\sigma \in \mathtt{NTEvol}(I)$ there do not exist s^T and t such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ and $s^T \in S_F$.

We say that I *passes* the set of tests $\mathcal{T}$ *for any time* if $\mathtt{pass}(I, \mathcal{T})$ and for all $(\sigma, t) \in \mathtt{TEvol}(I)$ such that $T' \stackrel{\sigma}{\Longrightarrow} s^{T'}$, for some $T' \in \mathcal{T}$, there exists $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ and $t = C(s^T)$.

We say that I *passes* the set of tests $\mathcal{T}$ *in the worst time* if $\mathtt{pass}(I, \mathcal{T})$ and for all $(\sigma, t) \in \mathtt{TEvol}(I)$ such that $T' \stackrel{\sigma}{\Longrightarrow} s^{T'}$, for some $T' \in \mathcal{T}$, there exists $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ and $t \leq C(s^T)$.

We say that I *passes* the set of tests $\mathcal{T}$ *in the best time* if $\mathtt{pass}(I, \mathcal{T})$ and for all $(\sigma, t) \in \mathtt{TEvol}(I)$ such that $T' \stackrel{\sigma}{\Longrightarrow} s^{T'}$, for some $T' \in \mathcal{T}$, we have that for all $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ it holds that $t \leq C(s^T)$. □

5 Test Derivation

In this section we present an algorithm to derive test cases from specifications. As usual, the basic idea underlying our algorithm consists in traversing the specification in order to get all the possible traces in an appropriate way. First, we introduce some additional notation.

Definition 6. Let $M = (S, I, O, T, s_{in}, \bar{y})$ be a TEFSM. We consider the following sets:

$$\mathtt{out}(s, i, \bar{x}) = \{o \mid \exists\, s', Q, Z, C :\ (s, s', i, o, Q, Z, C) \in T \ \wedge\ Q(\bar{x})\}$$

$$\mathtt{after}(s, i, o, \bar{x}, t) = \left\{ (s', \bar{x}', t + t') \;\middle|\; \begin{array}{r} \exists\, Q, Z, C : (s, s', i, o, Q, Z, C) \in T \wedge \\ Q(\bar{x}) \ \wedge\ Z(\bar{x}) = \bar{x}' \ \wedge\ C(\bar{x}) = t' \end{array} \right\}$$

□

The function $\mathtt{out}(s, i, \bar{x})$ computes the set of output actions associated with those transitions that can be executed from s after receiving the input i, and assuming that the value of the variables is given by $\bar{x}$. The function $\mathtt{after}(s, i, o, \bar{x}, t)$ computes all the *situations* that can be reached from a state s after receiving the input i, producing the output o, for a value of the variables $\bar{x}$, and after passing t units of time. By *situation* we mean triples denoting the reached state, the new value of the variables, and the cumulated time since the system started its performance.

The previously defined functions can be extended in the natural way to deal with sets:

$$\mathtt{out}(S, i) = \bigcup_{(s,\bar{x}) \in S} \mathtt{out}(s, i, \bar{x})$$

$$\mathtt{after}(D, i, o) = \bigcup_{(s,\bar{x},t) \in D} \mathtt{after}(s, i, o, \bar{x}, t)$$

Input: A specification $M = (S, I, O, Tran, s_{in}, \bar{y})$.
Output: A test case $T = (S', I, O \cup \{\texttt{null}\}, Tran', s_0, S_I, S_O, S_F, S_P, C)$.

Initialization:

- $S' := \{s_0\}, Tran' := S_I := S_O := S_F := S_P := C := \emptyset$.
- $S_{aux} := \{(\{(s_{in}, \bar{y}, 0)\}, s_0)\}$.

Inductive Cases: Choose one of the following two options until $S_{aux} = \emptyset$.

1. if $(D, s^T) \in S_{aux}$ then perform the following steps:
   (a) $S_{aux} := S_{aux} - \{(D, s^T)\}$.
   (b) $S_P := S_P \cup \{s^T\}$; $C(s^T) := \{t \mid (s, \overline{x}, t) \in D\}$.
2. If $S_{aux} = \{(D, s^T)\}$ and $\exists\, i \in I : \texttt{out}(S_M, i) \neq \emptyset$, with $S_M = \{(s, \bar{x}) \mid (s, \bar{x}, t) \in D\}$, then perform the following steps:
   (a) $S_{aux} := \emptyset$.
   (b) Choose i such that $\texttt{out}(S_M, i) \neq \emptyset$.
   (c) Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
   (d) $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $Tran' := Tran' \cup \{(s^T, i, s')\}$.
   (e) For all $o \notin \texttt{out}(S_M, i)$ do `{null is in this case}`
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $S_F := S_F \cup \{s''\}$; $Tran' := Tran' \cup \{(s', o, s'')\}$.
   (f) For all $o \in \texttt{out}(S_M, i)$ do
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $Tran' := Tran' \cup \{(s', o, s'')\}$.
      - $D' := \texttt{after}(D, i, o)$.
      - $S_{aux} := S_{aux} \cup \{(D', s'')\}$.

Fig. 3. Derivation of test cases from a specification.

The algorithm to derive tests from a specification is given in Figure 3. By considering the possible non-deterministic choices in the algorithm we may extract a full set of tests from the specification. For a given specification M, we denote this set of tests by $\texttt{tests}(M)$. Next we explain how our algorithm works. A set of *pending situations* D keeps those triples denoting the possible states, value of the variables, and time values that could appear in a state of the test whose definition, that is, its outgoing transitions, has not been yet completed. A pair $(D, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test and that the possible situations for that state are given by the set D. Let us remark that D is a set of situations, instead of a single one, due to the non-determinism that can appear in the specification.

Example 2. Let $M = (S, I, O, Tran, s_{in}, \bar{y})$ be a specification. Suppose that we have two transitions $(s, s', i, o, Q_1, Z_1, C_1), (s, s'', i, o, Q_2, Z_2, C_2) \in Tran$. If we want to compute the evolutions of M after performing i/o we have to consider both s' and s''. Formally, for a configuration $(s, \bar{x})$ and taking into account that the time elapsed so far equals t, we have to consider the set $\texttt{after}(\{(s, \bar{x}, t)\}, i, o)$.

The application of this function will return the different configurations, as well as the total elapsed time values, that could be obtained from $(s, \bar{x})$ and time t after receiving the input i and generating the output o. □

Following with the explanation of the algorithm, the set S_{aux} initially contains a tuple with the initial states (of both specification and test) and the initial situation of the process (that is, the initial state, the initial value of variables, and time 0). For each tuple belonging to S_{aux} we may choose one possibility. It is important to remark that the second step can be applied only when the set S_{aux} becomes singleton. So, our derived tests correspond to valid tests as given in Definition 4. The first possibility simply indicates that the state of the test becomes a passing state. The second possibility takes an input and generates a transition in the test labelled by this input. Then, the whole sets of outputs is considered. If the output is not expected by the implementation (step 2.(e) of the algorithm) then a transition leading to a failing state is created. This could be simulated by a single branch in the test, labelled by `else`, leading to a failing state (in the algorithm we suppose that *all* the possible outputs appear in the test). For the expected outputs (step 2.(f) of the algorithm) we create a transition with the corresponding output action and add the appropriate tuple to the set S_{aux}.

Finally, let us remark that finite test cases are constructed simply by considering a step where the second inductive case is not applied.

The next result relates, for a specification S and an implementation I, implementation relations and application of test suites. The non-timed aspects of our algorithm are based on the algorithm developed for the *ioco* relation. So, in spite of the differences, the non-timed part of the proof of our result is a simple adaptation of that in [Tre96]. Regarding temporal aspects, let us remark that the existence of different instances of the same timed evolution in the specification is the reason why only some tests (and for some time values) are forced to be passed by the implementation (e.g. sometimes we only need the *fastest/slowest* test). Specifically, we take those tests matching the requirements of the specific implementation relation. In this sense, the result holds because the temporal conditions required to conform to the specification and to pass the test suite are in fact the same.

Theorem 2. Let S, I be two `TEFSM`s. We have that:

- I $\texttt{conf}_a$ S iff I passes $\texttt{tests}(S)$ for any time.
- I $\texttt{conf}_w$ S iff I passes $\texttt{tests}(S)$ in the worst time.
- I $\texttt{conf}_b$ S iff I passes $\texttt{tests}(S)$ in the best time.

Proof. We will only prove the first of the results since the technique is similar for all of them.

First, let us show that I passes $\texttt{tests}(S)$ for any time implies I $\texttt{conf}_a$ S. We will use the contrapositive, that is, we will suppose that I $\texttt{conf}_a$ S does not hold and we will prove that I does not pass $\texttt{tests}(S)$ for any time. If I $\texttt{conf}_a$ S does not hold then we have two possibilities:

– Either $I\ \mathtt{conf}_{nt}\ S$ does not hold, or
– there exists a temporal evolution $(e,t) \in \mathtt{TEvol}(I)$ such that $e \in \mathtt{NTEvol}(S)$ and $(e,t) \notin \mathtt{TEvol}(S)$.

Let us consider the first case, that is, we suppose that $I\,\mathtt{conf}_{nt}\,S$ does not hold. Then, there exist two non-timed evolutions $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$ and $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$, with $r \geq 1$, such that $e \in \mathtt{NTEvol}(S)$, $e' \in \mathtt{NTEvol}(I)$, and $e' \notin \mathtt{NTEvol}(S)$. We have to show that if $e \in \mathtt{NTEvol}(S)$ then there exists a test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathtt{tests}(S)$ such that $T \stackrel{e}{\Longrightarrow} s^T$ and $s^T \in S_P$. We can construct this test by applying the algorithm given in Figure 3 and by resolving the non-deterministic choices in the following way:

for $1 \leq j \leq r$ **do**
- apply the second inductive case for the input action i_j
- apply the first inductive case for all the elements $(D, s^T) \in S_{aux}$ that have been obtained by processing an output different from o_j

endfor
apply first inductive case for the last (i.e. remaining) element $(D, s^T) \in S_{aux}$

The previous algorithm generates a test T such that $T \stackrel{e'}{\Longrightarrow} u^T$, with $u^T \in S_F$. This is so because the last application of the second inductive case for the output o'_r must be necessarily associated to the step 2.(e) since $e' \notin \mathtt{NTEvol}(S)$. Then, $I \parallel T \stackrel{e'}{\Longrightarrow}_t u^T$ for some time t. Given the fact that $T \in \mathtt{tests}(S)$ we deduce that $\mathtt{pass}(I, \mathtt{tests}(S))$ does not hold. Thus, we conclude I does not pass $\mathtt{tests}(S)$ for any time.

Let us suppose now that $I\,\mathtt{conf}_a\,S$ does not hold because there exists a temporal evolution $(e,t) \in \mathtt{TEvol}(I)$ such that $e \in \mathtt{NTEvol}(S)$ and $(e,t) \notin \mathtt{TEvol}(S)$. Let us consider the same test T that we defined before by taking into consideration the trace e. Since $e \in \mathtt{NTEvol}(S)$ we have that $T \stackrel{e}{\Longrightarrow} u^T$, with $s^T \in S_P$. Besides, since $(e,t) \in \mathtt{TEvol}(I)$, we also have $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$. The time stamps associated with the state s^T are generated by considering all the possible time values in which e could be performed in S. Thus, if $(e,t) \notin \mathtt{TEvol}(S)$ then $t \notin C(s^T)$. We conclude I does not pass $\mathtt{tests}(S)$ for any time.

Let us prove now that $I\,\mathtt{conf}_a\,S$ implies I passes $\mathtt{tests}(S)$ for any time. We will use again the contrapositive, that is, we will assume that I does not pass $\mathtt{tests}(S)$ for any time and we will conclude that $I\,\mathtt{conf}_a\,S$ does not hold. If I does not pass $\mathtt{tests}(S)$ for any time then we have two possibilities:

– Either $\mathtt{pass}(I, \mathtt{tests}(S))$ does not hold, or
– there exists $(e,t) \in \mathtt{TEvol}(I)$ and $T' \in \mathtt{tests}(S)$ such that $T' \stackrel{e}{\Longrightarrow} s^{T'}$ but there does not exist $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathtt{tests}(S)$ such that $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$, with $s^T \in S_P$ and $t \in C(s^T)$.

First, let us assume that I does not pass $\mathtt{tests}(S)$ for any time because $\mathtt{pass}(I, \mathtt{tests}(S))$ does not hold. This means that there exists a test $T \in \mathtt{tests}(S)$ such that there exist $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$, $s^T \in S_F$, and

t fulfilling $I \parallel T \stackrel{e'}{\Longrightarrow}_t s^T$. Then, we have $e' \in \mathtt{NTEvol}(I)$ and $T \stackrel{e'}{\Longrightarrow} s^T$. According to our derivation algorithm, a branch of a derived test leads to a fail state only if its associated output action is not expected in the specification. Thus, $e' \notin \mathtt{NTEvol}(S)$. Let us note that our algorithm allows to create fail state only as the result of the application of the second inductive case. One of the premises of this inductive case is $\mathtt{out}(S_M, i) \neq \emptyset$, that is, the specification is allowed to perform some output actions after the reception of the corresponding input. Thus, there exists an output action o_r and a trace $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$ such that $e \in \mathtt{NTEvol}(S)$. Given the fact that $e' \in \mathtt{NTEvol}(I)$, $e' \notin \mathtt{NTEvol}(S)$, and $e \in \mathtt{NTEvol}(S)$, we have that $I\, \mathtt{conf}_{nt}\, S$ does not hold. We conclude $I\, \mathtt{conf}_a\, S$ does not hold.

Let us suppose now that I does not pass $\mathtt{tests}(S)$ for any time because there exist $(e, t) \in \mathtt{TEvol}(I)$ and $T' \in \mathtt{tests}(S)$ such that $T' \stackrel{e}{\Longrightarrow} s^{T'}$ but there do not exist $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathtt{tests}(S)$ such that $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$, with $s^T \in S_P$ and $t \in C(s^T)$. We consider three possibilities. First, if $s^{T'}$ is a fail state then the considerations given in the previous paragraph can be applied. Thus, $I\, \mathtt{conf}_a\, S$ does not hold. Second, if the performance of the trace e in a test $T' \in \mathtt{tests}(S)$ reaches a state $s^{T'}$ that it is neither an acceptance or a fail state, then we can always find another test $T \in \mathtt{tests}(S)$ such that the performance of the sequence e reaches an acceptance state. Such a test T can be obtained by applying the first inductive case of the algorithm, instead of the second one, when dealing with the last input of the trace e. Finally, if $s^{T'}$ is an acceptance state then we simply consider $T' = T$. In the last two cases we obtain a test T such that $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$ and $s^T \in S_P$. Moreover, by taking into account our initial assumptions, we have $t \notin C(s^T)$. Since $s^T \in S_P$ we deduce $e \in \mathtt{NTEvol}(S)$. Besides, by considering that $t \notin C(s^T)$, we deduce $(e, t) \notin \mathtt{TEvol}(S)$. Finally, using that $(e, t) \in \mathtt{TEvol}(I)$, we conclude $I\ \mathtt{conf}_a\ S$ does not hold.

□

As a straightforward corollary we have that the dependencies between conformance relations that we presented in Theorem 1 also hold for the corresponding testing relations. For instance, since $\mathtt{conf}_a \Rightarrow \mathtt{conf}_w$ we also deduce that "passes X at any time" implies "passes X in the worst time."

6 Removing Restrictions on Implementations

If we allow implementations to present non-observable non-deterministic behavior then we may naturally introduce two more relations. The $\mathtt{conf}_{sw}$ relation requests that, for each of its evolutions, at least one instance of the implementation must be faster than the slowest instance, for the same evolution, of the specification. The $\mathtt{conf}_{sb}$ requests that, for each of its evolutions, at least one instance of the implementation is faster than the fastest instance of the specification.

Definition 7. Let S and I be two TEFSMs. We write $I\ \mathtt{conf}_{sw}\ S$ if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $(i_1/o_1,\ldots,i_r/o_r) \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ we have

$$\exists\, t_1, t_2 : \begin{pmatrix} ((i_1/o_1,\ldots,i_r/o_r), t_1) \in \mathtt{TEvol}(I) \wedge \\ ((i_1/o_1,\ldots,i_r/o_r), t_2) \in \mathtt{TEvol}(S) \wedge \\ t_1 \leq t_2 \end{pmatrix}$$

We write $I\ \mathtt{conf}_{sb}\ S$ if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $(i_1/o_1,\ldots,i_r/o_r) \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ we have

$$\exists\, t_1 : \begin{pmatrix} ((i_1/o_1,\ldots,i_r/o_r), t_1) \in \mathtt{TEvol}(I) \wedge \\ \forall\, ((i_1/o_1,\ldots,i_r/o_r), t_2) \in \mathtt{TEvol}(S) : t_1 \leq t_2 \end{pmatrix}$$

□

Next we generalize Lemma 1 to deal with the case where either the implementation or the specification (or both) are deterministic.

Lemma 2. Let I, S be two TEFSMs. We have the following results:

(1) If for all non-temporal evolution $(i_1/o_1,\ldots,i_r/o_r) \in \mathtt{NTEvol}(S)$ there do not exist two different time values t, t' such that $((i_1/o_1,\ldots,i_r/o_r), t)$ and $((i_1/o_1,\ldots,i_r/o_r), t') \in \mathtt{TEvol}(S)$, then $I\,\mathtt{conf}_w S$ iff $I\,\mathtt{conf}_b S$ and $I\,\mathtt{conf}_{sw} S$ iff $I\ \mathtt{conf}_{sb}\ S$.
(2) If for all non-temporal evolution $(i_1/o_1,\ldots,i_r/o_r) \in \mathtt{NTEvol}(I)$ there do not exist two different time values t, t' such that $((i_1/o_1,\ldots,i_r/o_r), t)$ and $((i_1/o_1,\ldots,i_r/o_r), t') \in \mathtt{TEvol}(I)$, then $I\,\mathtt{conf}_w S$ iff $I\,\mathtt{conf}_{sw} S$ and $I\,\mathtt{conf}_b S$ iff $I\ \mathtt{conf}_{sb}\ S$.
(3) If the conditions of the results (1) and (2) hold then the relations $\mathtt{conf}_w$, $\mathtt{conf}_b$, $\mathtt{conf}_{sw}$, and $\mathtt{conf}_{sb}$ coincide.

Proof. If the condition in (1) holds then the best instance of each evolution in the specification is actually the worst instance of that evolution. Similarly, if the condition in (2) holds then the best instance of each evolution in the implementation is the worst one as well. The last result is obtained from results (1) and (2) by applying transitivity between relations.

□

Let us note that the relation $\mathtt{conf}_a$ is different from the other relations even when the temporal behavior of both the specification and the implementation is deterministic. This is so because $\mathtt{conf}_a$ requires that time values in the implementation coincide with those in the specification, while other relations require that time values in the implementation are *less than or equal to* those of the specification.

Taking into account these new relations, we can extend Theorem 1 to include our five timed relations.

Theorem 3. The relations given in Definitions 3 and 7 are related as follows:

$$\begin{array}{ccc} I\ \mathtt{conf}_b\ S & \Rightarrow & I\ \mathtt{conf}_{sb}\ S \\ \Downarrow & & \Downarrow \\ I\ \mathtt{conf}_a\ S \Rightarrow I\ \mathtt{conf}_w\ S & \Rightarrow & I\ \mathtt{conf}_{sw}\ S \end{array}$$

Proof Sketch: The relation between $\mathtt{conf}_a$, $\mathtt{conf}_b S$, and $\mathtt{conf}_w$ was established in Theorem 1 and the proof can be found in [NR02]. Thus, we only need to consider $\mathtt{conf}_{sb}$ and $\mathtt{conf}_{sw}$. If $I\ \mathtt{conf}_w\ S$ then we know that each instance of a temporal evolution of I needs a time less than or equal to the one corresponding to the slowest instance, for the same evolution, of the specification S. In particular, there exists an instance fulfilling the condition imposed by $\mathtt{conf}_{sw}$. So, we conclude $I\ \mathtt{conf}_{sw}\ S$. The same reasoning can be also used to prove that $I\ \mathtt{conf}_b\ S$ implies $I\ \mathtt{conf}_{sb}\ S$.

Finally, we have to study the relation between $\mathtt{conf}_{sb}$ and $\mathtt{conf}_{sw}$. If $I\ \mathtt{conf}_{sb}\ S$ then we have that for any evolution of I there exists an instance being faster than the fastest instance of the same evolution in S. In particular, this instance is also faster than the slowest instance of S, so that we conclude $I\ \mathtt{conf}_{sw}\ S$. □

7 Conclusions and Future Work

We have presented a methodology for testing both functional and temporal aspects of systems where temporal behavior is critical. This requires us to endow tests with temporal requirements. Five implementation relations, differing in their temporal requirements, have been introduced and related. The nondeterminism of either the implementation or the specification induces some peculiarities in the testing methodology. In particular, when a test suite is applied to an implementation, the correctness of the temporal behavior is not assessed by checking the correctness of each test separately, but by checking temporal constraints over all the tests together. A sound and complete test derivation algorithm is constructed.

As future work we plan to improve the capability of our framework to express temporal constraints. In particular, we want to express conditions over the *minimal* time consumed by an action. Symbolic temporal constraints would allow to express both minimal and maximal bounds in a compact fashion. In addition, we plan to endow specifications with the capability to express temporal constraints over both actions and traces. In our current framework, temporal requirements are applied to traces. So, we assume that a low performance of an action may be compensated by other previous actions where performance was high. However, individual actions may have specific temporal requirements in some particular domains.

Acknowledgments. We would like to thank the anonymous referees for their helpful remarks.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BB04] L. Brandán Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *4th Int. Workshop on Formal Approaches to Testing of Software (FATES 2004), LNCS 3395*, pages 64–78. Springer, 2004.

[BSS86] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementations and their tests. In *Protocol Specification, Testing and Verification VI*, pages 349–360. North Holland, 1986.

[CL97] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems*, 1997.

[ED03] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *TestCom 2003, LNCS 2644*, pages 211–225. Springer, 2003.

[EDK02] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.

[HNTC99] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Workshop on Testing of Communicating Systems*, pages 197–214. Kluwer Academic Publishers, 1999.

[LMN04] K.G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In *4th Int. Workshop on Formal Approaches to Testing of Software (FATES 2004), LNCS 3395*, pages 79–94. Springer, 2004.

[MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.

[NR02] M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *FORTE 2002, LNCS 2529*, pages 1–16. Springer, 2002.

[SVD01] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

[Tre99] J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.