TESTING REAL-TIME PARAMETERIZED SYSTEMS

ARMELLE PRIGENT AND OLIVIER ROUX

Abstract. Testing is an important step in validation process of critical systems. Black box testing consists in verifying that the system conforms to the behaviour of its model. In the case of real-time systems, temporal constraints have to be taken into account in test generation. In this paper we tackle the problem of conformance testing for parameterized Real-Time systems. Parameterized timed automata allow to model systems with a finite set of parameters that represent unfixed clock constraints. The method proposed here generates parameterized test sequences associated with a set of constraints on parameters for which the specification do not have deadlock behaviours.

1. Introduction

The behaviour of a real-time system is highly dependent on the temporal performances of target hardware platforms used for the implementation. Moreover, reactions delays are constrained by temporal performances of the environment components which interact with the system. These constraints should be taken into account at the earliest stage of development process. Traditional specifications of this kind of systems take into account temporal constraints of environment or implementation with real timed values.

An approach described by [AHV93] consists in including parameters in the specification in order to describe temporal constraints that are unknown in the specification stage of the development process. In fact, a real-time constraint on the system makes sense only in a concrete environment. The model is represented by a parameterized timed automata network. Conformance testing increases confidence in software quality.

Traditional approaches of real-time test generation consist in computing test cases from specification with respect to real timed values in clocks constraints. In this paper, we tackle the problem of deriving test sequences from a specification where temporal constraints are expressed through parameters. We argue that parametric conformance testing can be helpful in system design process. In fact, during parametric test cases derivation the method leads to compute a set of constraints on the parameters.
These constraints can be helpful to choose suitable implementation components.

**Related works.** Several studies about timed test sequences generation have been proposed. An extension of test theory for mealy machines in the case of dense real time systems was proposed by Springintveld et al [SVD97]. This extension yields to a finite and complete set of tests but the authors admit that the method is highly exponential and is not usable in practice. Any approaches [CKL98, EDKE98, SPF01, Kon02] use a test purpose in order to select tests that have to be generated. A test purpose [JCTG96] - an abstract definition of an expected behaviour of the model - enables the user to choose behaviours which have to be analyzed and hence to reduce specification exploration. Nielsen and Skou [NS01] present timed approach of test cases generation from the well known Hennessy’s test theory [DH84]. They use symbolic representation of time [YPD94]. Test cases are computed by traversing forward and backward this representation. None of these approaches deals with parametric specification.

Parametric Model Checking problem has been introduced in [AHV93]. In this paper, it has been shown that the problem of computing a relevant set of parameters values that ensure a property correctness is undecidable in general (i.e. the emptiness problem). [HRSV01] proposed a semi-decision algorithm and a symbolic analysis method. A sub-class of parameterized automaton, the L/U automaton depicted in this approach, ensures algorithm termination. Annichini et al [AAB00] developed and implemented parametric analysis that allows non-linear parameters constraints.

Independently, both [AAB00] and [HRSV01] have used the same data structure to represent parametric constraints sets. The Parametric Difference Bound Matrix (PDBM) is an extension of Difference Bound Matrix described in [Dil89]. The parametric model-checking method proposed in [AAB00, HRSV01] produces the answer of property satisfaction together with a set of constraints on parameters for which the property holds.

**Our contribution.** Approaches described above concern model-checking parameterized systems. Conformance testing generation for parameterized real-time systems has not been explored yet. Traditional approaches of test generation methods provide test sequences with concrete temporal constraints to verify on the implementation under test. The main contribution of this paper holds in the proposition of a new parametric extension of conformance testing generation. Our proposition consists in producing parametric test sequences and a set of constraints on parameters that guarantee the specification leads to a final state.
In our approach, tested behaviours are specified using test purposes as in [CKL98, SPF01]. We use symbolic model checking techniques proposed by [HRSV01] to traverse the specification. Moreover, we propose indications on some solutions for parameters implementation before test execution. This method has been implemented in a tool called RTTG (Real Time Test Generator).

**Outline of the paper.** The organization of this paper is as follows: the second section deals with the formal specification of a real-time system with parameterized timed automaton. In this part, we present a simple example in order to apply our method. The third section describes the interest of specifying test with a test purpose and conformance relation. In this section, we present synchronous observation, and we apply it on the example. The parameterized symbolic sequences generation algorithm is depicted in section 5. The section 6 is about the concrete test case computation from symbolic sequences and we will discuss the problem of parameters implementation. In the last section, we present results obtained with our prototype RTTG.

## 2. Specification Outlines

Parametric real time systems are defined by networks of parameterized timed automata. In this paper, we will consider only deterministic specifications.

### 2.1. Parameterized Timed Automata

**Definition 1 (Linear Expressions).** Let us consider $P = \{p_1, \ldots, p_n\}$ a parameters set such that $p_i \in \mathbb{R}$ with $i \in \{1, \ldots, n\}$, $p$ a parameter in $P$ and $n \in \mathbb{N}$. $LE(P)$ is the set of linear expressions. $t \in LE(P)$ is defined according to the following grammar: $t ::= n | p_i | t - t | t + t | n \times t$

**Definition 2 (Clock Constraints).** Let $G(X)$ be the set of clocks constraints in the system, it is defined according to the following grammar: $g ::= x \sim e \mid g_1 \land g_2$ where $g_1, g_2$ are constraints, $x$ is a clock in $X$, $e \in LE(P)$ is a linear expression on the parameters and $\sim \in \{<, \leq, >, \geq\}$.

**Definition 3 (Parameterized Timed Automaton).** A parameterized timed automaton (PTA) is a tuple: $\mathcal{A} = (Q, q_0, Act, X, P, \rightarrow, I)$ where $Q$ is a finite set of locations, $q_0 \in Q$ is the initial location, $Act$ is the set of actions, $X$ is the set of clocks, and $P = \{p_1, \ldots, p_n\}$ is a finite set of parameters. $\rightarrow \subseteq Q \times Act \times G(X) \times 2^X \times Q$ is the transition relation. $(q, a, g, r, q') \in \rightarrow$ represents a switch from location $q$ to $q'$ with the action $a$. $g \in G(X)$ is a clock guard that specifies if the switch is enabled. $r \in 2^X$ is the set of clocks.
reset during the transition. The function $I : Q \rightarrow G(X)$ is a mapping (called invariant) that associates each location $q \in Q$ with a clock constraint in $G(X)$.

2.2. Specification model

The following example describes a system composed of 3 processes Sensor, Filter and Controller. This system is in charge to take measurements on the environment and apply filters on them. The Sensor process is aimed at collecting data from the environment and periodically sending them to the Filter process with signal $data!$. The Controller can send a data request to the Filter process. Filter process purpose is to collect data from the Sensor and to apply Filters on received data. Then filtered data (i.e. $Fdata$) are sent to the Controller. Then this process sends a data acknowledgment $ack!$. In order to express temporal constraints on this system, we take into account several parameters:

1. $l_{\text{min}}$ and $l_{\text{max}}$: respectively the minimal and maximal delay needed by the network to carry a signal from a process to another (i.e. latency)
2. $f$: the maximal delay before the arrival of another Sensor data (i.e. Sensor period)
3. $flt$: the delay needed by Filter process to filter a Sensor data

The automaton which models the Filter process of our example is depicted in figure 2.2. This model will be used in the whole paper as a running example.

3. Conformance Testing Requirements

3.1. Test Purpose Definition

A test purpose is a particular automaton representing a property to be checked on the implementation. The interest of using such an automaton during the test
sequences generation is that the behaviours explored on the model are limited and thus the state space explosion problem is reduced. A test purpose is an automaton with two kinds of final states: ACCEPT or REJECT. As described in [CKL98], the test purpose can also contain temporal constraints. The timed test purpose can have several clocks representing temporal constraints between two non-consecutive actions.

In the case of timed conformance testing, we argue that it is not necessary to use a timed test purpose. In fact, the aim of conformance testing is to determine if the implementation has the same behaviour than the specification. In the case of reachability property of the form "the system will lead to a specified state (within a time T)" , the test execution can conclude that the test purpose is not satisfied but the specification is respected (for example, the specification models a system where this specified state is reached within a time $T_s$). The verdict of test execution will lead to a timed inconclusive verdict. In order to check conformance of the implementation, we just need the timed constraints of specification. Temporal constraints on test purpose are thus not mandatory.

**Definition 4 (Test Purpose).** A test purpose is an automaton:

$$TP = (Q_{TP}, q_0^{TP}, Act_{TP}, \rightarrow_{TP}, \text{ACCEPT}^{TP}, \text{REJECT}^{TP})$$

with $Q_{TP}$ a finite set of control states, $q_0^{TP}$ the initial control state, $Act_{TP}$ a set of actions, and $\rightarrow_{TP}$ the transition relation. The two sets of end-states are denoted $\text{ACCEPT}^{TP}$ and $\text{REJECT}^{TP}$. 

WTRTES 2003
Let $S$ be the specification automata, the alphabet recognized by the test purpose is the same as the alphabet of the specification (i.e. $Act_{tp} = Act_s$). In order to recognize each run from the specification, the test purpose has to be complete (i.e. $\forall q \in Q_{tp}$ and $\forall a \in Act_{tp}$, $\exists (q, a, q') \in \rightarrow_{tp}$). We use $\ast$ to depict all actions except the other ones explicitly enabled from $q$.

If the test purpose is complete, it will recognize all executions of specification and we are sure that the synchronous product between specification and test purpose will keep all the behaviours of the specification. The REJECT set contains states associated with behaviours which are not taken into account. Each behaviour in the REJECT set leads to a REJECT state in the synchronous product. The behaviours leading to REJECT state will not be taken into account in reachability analysis. This set reduces the specification exploration. This test purpose definition is similar to the one described in the TGV method [Mor00].

Example 3.1 (Test purpose for the filter process). The figure 3.3 describes an example of test purpose for the Filter process depicted in figure 2.2. This test purpose is aimed at guaranteeing that a data request from the Controller will eventually lead to a filtered data sending. In this case, we do not take into account behaviours when the data filtering is interrupted by a Sensor data reception.

3.2. Conformance Relation

Conformance testing aims at verifying that external behaviours of implementation conform to external behaviours of the model. Let us consider $Tr(S)$ and

Figure 3.3: The test purpose process
Testing Real-Time Parameterized Systems

39

Tr(I) the set of observable traces of specification and implementation, the conformance relation can be expressed as follows: ∀σ ∈ Tr(I) imply σ ∈ Tr(S) σ is a trace in observable behaviours of implementation and specification. Each trace in the observable behaviour of the implementation I is a trace of the observable behaviour of the specification S.

3.3. Synchronous Observation of Specification

The test generation method is based on a depth first analysis on synchronous product between the specification and the automaton representing the test purpose. The synchronous product is computed as described in [ACD90]. Let $S = (Q_s, q_0^s, Act_s, X_s, P_s, \rightarrow_s, I_s)$ and $TP = (Q_{tp}, q_0^{tp}, Act_{tp}, \rightarrow_{tp})$ be respectively the specification parametric timed automata and the test purpose, the synchronous product of these automata, denoted $PS = S \times TP$, is the parameterized timed automata:

$$PS = (Q_s \times Q_{tp}, Act_s \cup Act_{tp}, X_s, \rightarrow_{ps}, I_{PS})$$

The transition relation is defined by the following rule:

$\rightarrow_{ps}$ and $a \in Act_s \cap Act_{tp}$, $(q_s, q_{tp}), a, q_s, r_s, (q_s', q_{tp}') > \in \rightarrow_{ps}$.

The invariant relation is $I_{PS}(q_s, q_{tp}) = I(q_s)$. Actually, the test purpose is an untimed automaton, the same as the specification. REJECT and ACCEPT sets are managed according to the following rules:

$$\text{ACCEPT}^{PS} = \{(q_s, q_{tp}) | (q_s, q_{tp}) \in Q_{PS} \land q_{tp} \in \text{ACCEPT}^{TP}\}$$

$$\text{REJECT}^{PS} = \{(q_s, q_{tp}) | (q_s, q_{tp}) \in Q_{PS} \land q_{tp} \in \text{REJECT}^{TP}\}$$

In order to reduce execution time, the synchronous product between test purpose and processes of system is computed at the first step of test generation. This means that global system (i.e. composition of all processes of the system) will never be computed. In order to obtain a better reduction of execution times, the synchronous product could be computed on the fly during the reachability analysis step.

3.4. Synchronous Product for the example

The Figure 3.4 shows the synchronous product between the Filter process model and the test purpose depicted in Figure 3.3.

4. Symbolic Test Generation

The forward reachability analysis algorithm is applied on the synchronous product computed from the model and the test purpose. This algorithm is an
adaptation from the one in [HRSV01]. In fact, the original algorithm checks that property is correct during the system reachability analysis. In this case, checking the property in the states is not necessary. We only compute reachable states of synchronous product and a set of constraints on parameters.

### 4.1. Symbolic States

A symbolic state is a tuple \((q, C, D)\) where \(q\) is the location, \(C\) is a set of constraints on parameters, and \(D\) a set of constraints on clocks values. Let \(S'\) represent the set of symbolic successors \(\{(q', C', D')\}\) for a symbolic state \((q, C, D)\). Operations on the set of constraints are computed using the \text{PDBM} (Parametric Difference Bounded Matrices) that we introduce now.

WTRTES 2003
4.2. Parametric Difference Bound Matrix

The definition of PDBM described here is extracted from [HRSV01, AAB00]. A more precise definition of operators for this data structure can be found in these papers. The main difference between DBM and PDBM is that elements in the parametric case, elements in the matrix are linear expressions on parameters.

Definition 5. Parametric Difference Bound Matrix Let us consider the set of clocks \( X = \{x_1, \ldots, x_n\} \). A parametric difference bound matrix is a \((n+1) \times (n+1)\) matrix. Each element \(D_{ij}\) of the matrix is a pair \((e_{ij}, \prec)\) that represents a constraint of the form a simple constraint \(x_i - x_j \prec e_{ij}\), with \(e_{ij} \in LE(P)\) and \(\prec \in \{<, \leq\}\). For constraints of the form \(x_i \prec e_{i0}\) a special clock \(x_0\) which value is 0 is inserted and this constraint is represented with \(x_i - x_0 \prec e_{i0}\). For all \(i\), \(D_{ii}\) represents \(x_i \leq x_i\).

Example 4.1 (An example of PDBM). The Figure 4.5 shows an example of PDBM.

\[
\begin{bmatrix}
  0 & x & y \\
  0 & (0, \leq) & (-1, \leq) & (-p, <) \\
  x & (q, <) & (0, \leq) & (0, \leq) \\
  y & (r, \leq) & (0, \leq) & (0, \leq)
\end{bmatrix}
\]

Figure 4.5: PDBM for the parametric constraint \(1 \leq x < q \land p < y \leq r\)

A constrained PDBM is a pair \((C, D)\) where \(C\) is a set of constraints on parameters and \(D\) a PDBM.

4.2.1. Operations on PDBM

The symbolic successor computation uses some crucial operations on PDBM described in [HRSV01, AAB00]. We will describe these operations in this paragraph in a different manner.

Clock reset. Let us consider \((C, D)\) a constrained PDBM and \(\{x_i := 0\}\) a clock reset, as in non-parametric case the PDBM after clock reset is computed by replacing:

- Each bound \(D_{ij}\) with \((0, \leq)\)
- Each bound \(D_{ij}\) with \((0, \leq)\)
- Each bound \(D_{ij}\) with \(D_{0j}\) and \(D_{ji}\) with \(D_{0i}\)
Timed successor. As in non parametric case, each bound \( D_{ij} \) is changed to \((\infty, <)\).

Conjunction with a parametric constraint. In the non-parametric case, the conjunction with a constraint consists in keeping the smallest bound of the two constraints. In a parametric timed automata, the conjunction operation can result in several symbolic states. For example, \( \{ x \leq q \} \land \{ x \leq p \} \) depends on \( p < q \) (i.e. \( x \leq q \) if \( q < p \) and \( x \leq p \) if \( p \leq q \)). Let \( (C, D) \) be a constrained PDBM and \( g \) be a parametric constraint. First, before adding a constraint, we have to define if the bound of the PDBM is smaller than the corresponding bound in parametric constraint. \( C \) is the operator that construct a constraint that states bound of the PDBM is smaller than the bound define by the parametric constraint. Let \( D_{ij} = (e_{ij}, \preceq_{ij}) \) be a constraint in the PDBM, \( \preceq \) is the smallest of the two comparison operators \( \{ \preceq_{ij}, \preceq_g \} \):

\[
C(D, x_i - x_j \preceq_g e_g) = e_{ij} \prec e_g
\]

The second step consists in verifying if the constraint produced by \( C \) is consistent with the set of constraint on parameters \( C \). \( O \) is the function that answer this question:

\[
O(c, C) = \begin{cases} 
  yes & \text{if } C \models c \\
  no & \text{if } C \models \lnot c \\
  split & \text{Else}
\end{cases}
\]

Using these two operators, one can define parametric conjunction \((C, D) \land g\). The result of this operation is the set of symbolic states computed according to the following rules:

- If \( O(C(D, g), C) = yes \) then \( \{ (C', D') \} = (C, D) \)
- If \( O(C(D, g), C) = no \) then \( \{ (C', D') \} = (C, D[g]) \)
- If \( O(C(D, g), C) = split \) then
  \[
  \{ (C', D') \} = \{ (C', D') \} \cup \{ (C \cup \{C(D, g)\}, D) \}
  \]
  \[
  \{ (C', D') \} = \{ (C', D') \} \cup \{ (C \cup \{C(D, g)\}, D[g]) \}
  \]

\( D[g] \) denotes the PDBM obtained after changing the bound in \( D \) with bound of parametric constraint \( g \). For example, let \( (C, D) \) be a constrained PDBM. \( C = \{ a < b \} \) and \( D \) represents the set of constraints \( \{ 0 < x < d \} \). The conjunction of \((C, D)\) with the parametric constraint \( x < e \) produce the following constrained PDBM:

- \( (C_1, D_1), C_1 = \{ a < b, c < d \} \) and \( D_2 \) contains \( x < e \)
- \( (C_2, D_2) \), \( C_2 = \{ a < b, d \leq c \} \) and \( D_2 \) contains \( x < d \)

4.3. Parametric Constraints Construction

During the forward reachability analysis, we construct a set of constraints on parameters that guarantee that the specification will not present deadlocks due
to the parameters. The forward reachability algorithm constructs constraints

\[
x > p \\
x \leq q
\]

(a) Deadlock if \( p > q \)

\[
x > n \\
x \leq m
\]

(b) Deadlock if \( n > m \)

Figure 4.6: Two examples of deadlocks due to parameters

on parameters according to the following rules:

(a) \( \forall l, l' \in Q \) and \( p, q \in P \) as \( l \overset{a; g; r}{\rightarrow} l' \), \( x_i > p \) and \( x_i < q \in I(l) \), then \( C_l = C_l \cup \{ p \leq q \} \).

(b) \( \forall l, l' \in Q \) and \( p, q \in P \) as \( l \overset{a; g; r}{\rightarrow} l' \), \( x_i > p \) and \( x_i < q \in I(l') \)

and \( x_i := 0 \notin r \), then \( C_l = C_l \cup \{ p \leq q \} \).

4.4. Symbolic Successor Computation

According to the symbolic semantics of parameterized timed automata, the set \( S' \) of symbolic successors is computed in using the following method: For all transition \( q \overset{a; g; r}{\rightarrow} q' \):

1. Apply constraint on parameters for the guard: \( \forall x_i \ni p \in g \) and \( x_i < q \in I(q) \), then: if \( (C_q \cup \{ p \leq q \}) \neq \emptyset \), \( S'1 = (C_q \cup \{ p \leq q \}, D_q) \).

2. Apply the guard of the transition: \( \forall (C_i, D_i) \in S'_1 \), \( (C'_i, D'_i) = (C_i, D_i) \land g \), \( S'_2 = S'_2 \cup (C'_i, D'_i) \).

3. Reset clocks in the reset set of the transition: \( \forall (C_i, D_i) \in S'_2 \), \( S'_3 = \cup (C_i, D_i[x \backslash 0]) \forall x \in R \).

4. Let time elapsing in the reached state: \( \forall (C_i, D_i) \in S'_3 \), \( S'_4 = \cup (C_i, D_i) \).

5. Apply constraint on parameters for the invariant: \( \forall x_i \ni p \in g \), \( x_i < q \in I(q') \) and \( x_i := 0 \notin r \), then: if \( (C_i \cup \{ p \leq q \}) \neq \emptyset \), \( \forall (C_i, D_i) \in S'_4 \), \( S'5 = (C_i \cup \{ p \leq q \}, D_i) \).

6. Apply the target state invariant: \( \forall (C_i, D_i) \in S'_5 \), \( (C'_i, D'_i) = (C_i, D_i) \land I_q' \), \( S'_6 = S'_6 \cup (C'_i, D'_i) \).

After last step, \( \forall (C', D') \in S'_6 \), we can write: \( (q, C, D) \rightarrow (q', C', D') \).
5. Parameterized Tests Construction

In this section, we tackle the problem of generating symbolic test sequences using symbolic successor computation. The symbolic sequences construction is based on the well-known forward reachability algorithm described by [Pet99].

5.1. Forward Reachability Algorithm

This algorithm is similar to the one used for non-parameterized timed automata, except that this one computes constraints on clocks and constraints on parameters during graph traversal.

In order to save states, this algorithm depicted in figure 5.1 manipulates two sets of states. PASSED stands for the reached states and WAITING contains the unexplored states. The analysis starts at the initial state and computes the symbolic reachable states in a step from an existing symbolic state already encountered. When the algorithm ends, the PASSED list contains the reachable states space. This algorithm is similar to the one used for non-parameterized timed automata, except that this one computes constraints on clocks and constraints on parameters. The algorithm is the following:

5.2. Symbolic Sequence for the Filter process

This section deals with the results obtained when applying the forward reachability algorithm to the example presented in section 2.2. One of the symbolic sequences obtained by reachability analysis on the synchronous product is depicted in figure 5.7. Here, we only consider, sequences leading to an accepting state. The set of constraints $C$ on parameters obtained from this analysis is the union of constraints on each state in the symbolic sequence: $C = \cap C_i \forall q_i$

At this point of the test generation, we already know that the symbolic sequence will always terminate in a final state if the set of constraints on parameters is achieved.

In this example, the set of constraints $C = \{flt + l_{min} < f ; 2 + l_{min} < f\}$ guarantees that the ACCEPT state will always be reached. Those results have the following meaning:

- The first result $flt + l_{min} < f$ means that the maximal delay before the Sensor sends data must be lower than the time needed by Filter to process the previous data and send it to the Controller process.
- Finally, the forward reachability shows that for this symbolic sequence $2 + l_{min} < f$. This means that the Sensor period depends on the network latency and more precisely that it must be greater than the maximal latency.
Testing Real-Time Parameterized Systems

Fonction 5.1 Function that computes parametric symbolic traces

Function SymbolicTracesComputation(ParametricTimedAutomata PS):
vector of traces
2: PASSED:=\emptyset
PASSED:=\emptyset
WAITING :=\{L_0 = (q_0, \emptyset, D)\}
TRACES:= \emptyset
4: While WAITING \neq \emptyset Do
WAITING:= WAITING - \{(Q, C_p, D)\}
6: TRACETEMP := \emptyset
For all T \in Traces of the form (l_0, D_0), \ldots, (l, D) Do
8: TRACETEMP := TRACETEMP \cup T
TRACES := TRACES - T
10: End While
12: If \exists\{l, D\} \in PASSED then
PASSED := PASSED \cup \{(Q, C_p, D)\}
For all P', a, g, r as \{(Q', C_{p'}, D')\} \xrightarrow{a,g,r} \{(Q, C_p, D)\}
and D' = ((r(q \land C))') and C_{p'} \neq \emptyset Do
14: WAITING := WAITING \cup \{(Q', C_{p'}, D')\}
T := T \cup (l', D')
16: End For
End If
18: TRACES := TRACES \cup T
End While

latency plus the needed delay for Controller process to acknowledge the data.
ACCEPT state is reachable from state S_3 \cdot O_1 provided f < flt + l_{max} or flt + l_{max} \leq f. These two cases have to be taken into account. As an example for test case concretization, we choose to describe the case flt + l_{max} \leq f.

6. Symbolic Sequences Concretization

After symbolic sequences computation (see the above section), it is necessary to define for each of these sequences, one or more concrete test sequence. These ones are the test cases actually executed on the implementation.
First step. The first step for the test case concretization is to specify which signals are controlled by the tester. In the case of the example depicted above, the tester can be substituted to Controller process. Therefore, signals get?
and $ack$? are sent by the tester to the specification. As the test case behaves as an observer for the implementation under test, signals on the concrete test sequence are dual of those on the specification. For the transition, $q \xrightarrow{get} q'$ the corresponding transition on the test case is $q \xrightarrow{get} q'$ and so on for signal $ack$.

**Second step.** The aim of conformance testing is to provide verdicts on the conformance between the implementation under test and its initial specification. Verdicts have the following meaning:

- **PASS:** The behaviour of the implementation conforms to the specification’s one.

Figure 5.7: A symbolic sequence computed from synchronous product
FAIL: The implementation does not behave like the specification.

Symbolic sequences which end-states are labelled by ACCEPT design sequences leading to a successful system state. All ACCEPT states are moved in PASS states which correspond to a successful test verdict. Sequences leading to a REJECT state are not taken into account. They represent behaviours we do not want to test.

Third step. Symbolic states in the sequence are of the form \((q_i, C_i, D_i)\). The set of clocks constraints reachable in each state is represented by matrix \(D_i\). When a transition occurs, the tester has to verify that this set of constraints is achieved. Let \(D\) be the current temporal constraints during test execution, \(q\) the current location, and \(I_q\) be the \(q\) location invariant, if \(D \land I_q \notin D_q\), the tester must return a FAIL verdict.

![Test Sequence after the last step of test concretization](image)

Last Step. In the case a transition switch of the test sequence is enabled by a signal occurrence, the timed tester has to choose the instant when the signal must be sent. Let us consider the example, the signal \(get!\) has to be sent to

WTRTES 2003
the Filter in a delay that respects some constraints. In order to specify the signal emit instant, a new clock \( z \) is added in the test sequence. This clock has to be reset in the transition that precedes the occurrence of signal sending. Then, adding a guard of the form \( z == t \) with \( t \) an unknown parameter, will guarantee that the transition will be passed at the fitting instant. The first thing is to guarantee that the sending instant respects constraints in the current state. The second step is to ensure that the final state could be reached. The possible values of \( t \) are computed using the backward constraints propagation method developed by [Nie00]. This method consists in computing for each symbolic state a strengthened zone \( Z \). The strengthened zone is the set of constraints in the state which ensure the final state will be reachable from the initial state. This set of constraints for each state is computed with a method that is similar to the backward reachability analysis. The method starts at the end state of the symbolic sequence and computes iteratively the symbolic predecessor. Each state is associated with a strengthened zone \( Z_i \). More precise details on backward constraints propagation method can be found in [Nie00]. The figure 6.8 depicts the test sequence obtained after the steps explained above. At this step, the test sequence is the one described in figure 6.8. The selection of the signal emitting instant can be done inside \( Z_i \). The last step is to choose a delay among these possible represented by \( Z_i \).[Nie00] proposes three strategies for the choice of a value in \( Z_i \):

- The choice of the smallest delay checks promptness of implementation.
- The choice of a delay in the middle checks persistence of implementation.
- The choice of the largest delay checks the patience of the implementation.

In our running example, we choose to send signal as earliest as possible: \( z == \text{lmin} \).

6.1. Test Sequences Instanciation with Parameters Implementation

Dealing with indeterminism. A problem that occurs when manipulating parameterized specification is the indeterminism that appears because of the parameters. This indeterminism lasts only during the test construction phase. There is no more indeterminism when the parameters have been instanciated.

Parameters Implementation. At this stage of the test generation process, test cases may be applied on the implementation except that actual values have to be assigned to parameters. Two solutions are available. First, parameters can be instanciated via human choice. Every parameters values can be computed or evaluated by those who execute the test sequences. Another approach is
to supply the tester with uninstanciated test sequences and leave to it the task of evaluating parameters values directly on the implementation. This solution needs a particular architecture for the tester. There must be a process able to evaluate physical characteristics corresponding to parameters on the specification. After choosing a value for parameters instances, the test sequences that respects these values have to be executed on IUT, in order to state on the conformance. The conformance is established if none of the test executions produces a FAIL verdict.

**Real Test Case for Filter process.** Let us suppose that after an evaluation of implementation, the network latency, period and filtering delay have been fixed to the following values:

1. \( l_{\text{max}} = 2 \) and \( l_{\text{min}} = 1 \)
2. \( \text{flt} = 1 \)
3. \( f = 5 \)

The figure 6.9 depicts a concrete test case for the Filter process.

Another approach, on this example could have been to choose among several physical sensors in which one will satisfy this sequence after the evaluation of the needed filtering delay and the network latency.

Figure 6.9: Real test sequence for Filter process
7. Experiments and Results

The method described above has been implemented in a tool called Real Time Test Generator (RTTG). In this section, we show some results of experiments achieved using RTTG. We present here the results of three cases studies from which we have generated some parameterized symbolic test sequences.

The table 1 gives the number of transitions in each system, the number of parameters that have been set in the system, and the number of symbolic tests generated for each example. The fourth column gives the number of constraints on parameters that have been computed during symbolic test sequences derivation process. Finally, the last column of the table gives the time (in seconds) needed by the tool RTTG to generate the set of symbolic tests.

The first example is the well known railway crossing problem depicted in [Alu99]. On this example, we have parameterized the time needed by the door to react when receiving the order to go up. Moreover, the delay for this door to pass from down state to up state is represented by another parameter.

The second example is a small system composed of 3 processes Sensor, Filter and Controller. This system is in charge of taking measurements on the environment and of applying filters on them. One of the parameters added in the system is the delay needed by Filter process to filter a Sensor data.

Finally, we have used test generation on a more genuine case study which is a part of an avionic system presented in [JE02]: the ailerons control system. This system controls ailerons of an aircraft. For example, in the ailerons control system, we consider parameters $d_1$ and $d_2$ that represent the minimal and maximal delays of two communication buses. In these examples, the input format is an textual UPPAAL file on which clocks constraints expressed with constants have been parameterized by hand in the model.

<table>
<thead>
<tr>
<th></th>
<th># transitions</th>
<th># parameters</th>
<th># symbolic tests</th>
<th># parameters constraints</th>
<th>execut. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Railway cross.</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Filter-Sensor</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Ailerons Cont.</td>
<td>4451</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 1: Results of RTTG’s Experiments

Comments On the Results of the Railway Crossing. As an example, the test derivation process for the railway crossing problem produces in less than 2 seconds 2 symbolic tests associated with 3 constraints on parameters. One of
them states that the time needed by the door process to react when receiving the order to close must be less than the time it needs to close.

8. Conclusions and Future Works

We have presented a method to derive test sequences from a parameterized real-time model. The test specification consists in representing behaviours that have to be tested with a test purpose. The test generation uses symbolic representation of temporal constraints. The main difference with classical timed test generation is the computation of constraints on parameters values during forward reachability analysis. These constraints on parameters can help the designer in the choice of system components. We have introduced here the prototype RTTG that implement this method. Another interest of parameterized test generation for real-time systems holds in real-time test execution architecture. Indeed, the delay of the network between the tester and the IUT could be taken into account by parameters in specification.

References


Authors addresses:

Armelle Prigent.
L3i
IUT Informatique de La Rochelle
15 Rue Francois de Vaux de Foletier
17000 LA ROCHELLE
aprigent@univ-lr.fr

Olivier Roux.
IRCCYN
Ecole Centrale de Nantes
1, rue de la Noé
BP 92 101
44321 Nantes CEDEX 03
Olivier.Roux@irccyn.ec-nantes.fr

WTRTES 2003