Controllable test cases for the distributed test architecture*

Robert M. Hierons[1], Mercedes G. Merayo[1,2], and Manuel Núñez[2]

[1] Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom,
rob.hierons@brunel.ac.uk, mgmerayo@fdi.ucm.es
[2] Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain,
mn@sip.ucm.es

Abstract. In the distributed test architecture, a system with multiple ports is tested using a tester at each port/interface, these testers cannot communicate with one another and there is no global clock. Recent work has defined an implementation relation for testing against an input-output transition system in the distributed test architecture. However, this framework placed no restrictions on the test cases and, in particular, allowed them to produce some kind of nondeterminism. In addition, it did not consider the test generation problem. This paper explores the class of controllable test cases for the distributed test architecture, defining a new implementation relation and a test generation algorithm.

1 Introduction

If the system under test (SUT) has physically distributed interfaces/ports then it is normal to place a tester at each port. If testing is black-box, there is no global clock, and the testers cannot directly communicate with each other then we are testing in the distributed test architecture, which has been standardised by the ISO [1]. The use of the distributed test architecture reduces test effectiveness (see, for example, [2–5]).

The area of testing against a state-based model has received much attention [6–9]. The main advantage of using a formal approach is that many testing processes can be automated (see [10] for a discussion on the advantages of formal testing and [11] for a survey). However, most previous work on testing in the distributed test architecture has considered deterministic finite state machine (DFSM). The IOTS formalism is more general: in a DFSM input and output alternate and DFSMs have a finite state structure and are deterministic. The last restriction is particularly problematic since distributed systems are often nondeterministic. While the implementation relation **ioco** [9], that is usually used in testing from an IOTS, has been adapted in a number of ways (see, for

* Research partially supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and the Marie Curie project MRTN-CT-2003-505121/TAROT.

example, [12–19]), only recently has the problem of testing from an IOTS in the distributed test architecture been investigated [20].This work introduced an implementation relation **dioco** but this assumes that any deterministic test case can be applied including test cases that are not *controllable*. Controllable test cases are such that there does not exist a situation where a local tester has observed a trace after which either it should apply an input or wait for output, depending on what has happened at the other port. The problem is that in such situations local testers do not know when they have to apply their input.

This paper defines what it means for a test case to be controllable and shows that we can decide in polynomial time whether a test case has this property. We define a new implementation relation for controllable testing in the distributed test architecture. Finally, we give an algorithm for generating these test cases. This paper therefore extends the work of [20] by considering controllable testing. In addition, it is the first paper to give a test generation algorithm for testing against an IOTS in the distributed test architecture.

2 Preliminaries

This section defines input-output transition systems and associated notation and outlines the distributed test architecture.

2.1 Input output transition systems

We use *input-output transition systems* to describe systems. These are labelled transition systems in which we distinguish between inputs and outputs [9].

Definition 1. *An* input-output transition system *s (*IOTS*) is defined by* (Q, I, O, T, q_{in}) *in which* Q *is a countable set of states,* $q_{in} \in Q$ *is the initial state,* I *is a countable set of inputs,* O *is a countable set of outputs, and* $T \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$*, is the transition relation, where* τ *represents an internal (unobservable) action. A transition* (q, a, q') *means that from state* q *it is possible to move to state* q' *with action* $a \in I \cup O \cup \{\tau\}$*. We let* $\mathcal{IOTS}(I, O)$ *denote the set of IOTSs with input set* I *and output set* O*.*

State $q \in Q$ *is* quiescent *if from* q *it is not possible to produce output without first receiving input. We can extend* T *to* T_δ *by adding* (q, δ, q) *for each quiescent state* q*. We let* $\mathcal{A}ct = I \cup O \cup \{\delta\}$ *denote the set of observable actions and so* $\tau \notin \mathcal{A}ct$*. Process* s *is* input-enabled *if for all* $q \in Q$ *and* $?i \in I$ *there exists* $q' \in Q$ *such that* $(q, ?i, q') \in T$*.* s *is* output-divergent *if it can reach a state in which there is an infinite path that contains outputs and internal actions only.*

Given action a *and process* s*,* $a.s$ *denotes the process that performs* a *and then becomes* s*. Given a countable set* S *of processes,* $\sum S$ *denotes the process that can nondeterministically choose to be any one of the processes in* S*. We will sometimes use the binary operator* $+$ *to denote the election between processes.*

Processes and states are effectively the same since we can identify a process with its initial state and we can define a process corresponding to a state q of s

by making q the initial state. Thus, we use states and process and their notation interchangeably.

We use the normal notation in which we precede the name of an input by ? and the name of an output by !. We assume that all processes are input-enabled and are not output-divergent. The intuition behind the first restriction is that systems should be able to respond to any signal received from the environment. Regarding the second restriction, in the distributed testing architecture quiescent states can be used to combine the traces observed at each port and reach a verdict. If a process is output-divergent then it can go through an infinite sequence of non-quiescent states, so that local traces cannot be combined.

Traces are sequences of visible actions, possibly including quiescence, and are often called *suspension traces*. Since they are the only type of trace we consider, we call them *traces*. The following is standard notation in the context of **ioco**.

Definition 2. *Let $s = (Q, I, O, T, q_{in})$ be an IOTS.*

1. *If $(q, a, q') \in T_\delta$, for $a \in \mathcal{A}ct \cup \{\tau\}$, then we write $q \xrightarrow{a} q'$.*
2. *We write $q \stackrel{a}{\Longrightarrow} q'$, for $a \in \mathcal{A}ct$, if there exist $q_0, \ldots, q_m$ and $k \geq 0$ such that $q = q_0$, $q' = q_m$, $q_0 \xrightarrow{\tau} q_1, \ldots, q_{k-1} \xrightarrow{\tau} q_k$, $q_k \xrightarrow{a} q_{k+1}$, $q_{k+1} \xrightarrow{\tau} q_{k+2}, \ldots, q_{m-1} \xrightarrow{\tau} q_m$.*
3. *We write $q \stackrel{\epsilon}{\Longrightarrow} q'$ if there exist $q_1, \ldots, q_k$, for $k \geq 1$, such that $q = q_1$, $q' = q_k$, $q_1 \xrightarrow{\tau} q_2, \ldots, q_{k-1} \xrightarrow{\tau} q_k$.*
4. *We write $q \stackrel{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1 \ldots a_m \in \mathcal{A}ct^*$ if there exist $q_0, \ldots, q_m$, $q = q_0$, $q' = q_m$ such that for all $0 \leq i < m$ we have that $q_i \stackrel{a_{i+1}}{\Longrightarrow} q_{i+1}$.*
5. *We write $s \stackrel{\sigma}{\Longrightarrow}$ if there exists q' such that $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ and we say that σ is a* trace *of s. We let $\mathcal{T}r(s)$ denote the set of traces of s.*

Let $q \in Q$ and $\sigma \in \mathcal{A}ct^*$ be a trace. We consider

1. *q **after** $\sigma = \{q' \in Q | q \stackrel{\sigma}{\Longrightarrow} q'\}$*
2. *$out(q) = \{!o \in O | q \stackrel{!o}{\Longrightarrow}\}$*
3. *Given a set $Q' \subseteq Q$, we consider that Q' **after** $\sigma = \cup_{q \in Q'} q$ **after** σ and $out(Q') = \cup_{q \in Q'} out(q)$.*

Process s is deterministic *if for all $\sigma \in \mathcal{A}ct^*$, $|out(q_{in}$ **after** $\sigma)| \leq 1$.*

In testing from a single-port IOTS it is usual to use **ioco** [9, 21].

Definition 3. *Given $s, i \in \mathcal{IOTS}(I, O)$ we write i **ioco** s if for every trace $\sigma \in \mathcal{T}r(s)$ we have that $out(i$ **after** $\sigma) \subseteq out(s$ **after** $\sigma)$.*

2.2 Multi-port input-output transition systems

The two standard (ISO) test architectures are shown in Figure 1. In the local test architecture a global tester interacts with all of the ports of the SUT. In the distributed test architecture there is a local tester at each port [1]. For the sake of simplicity, in this paper we will sometimes assume that there are only

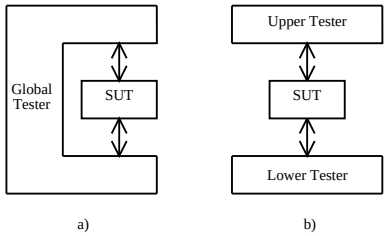

Fig. 1. The local and distributed test architectures.

two ports which we call U and L. However, all the results can be easily extended to $n > 2$ ports. We use the term IOTS where there are multiple ports and when there is only one port we use the term single-port IOTS.

For an IOTS (Q, I, O, T, q_{in}) with port set $\mathcal{P} = \{U, L\}$ we partition I into sets I_U and I_L of inputs that can be received at U and L respectively. Similarly, O can be partitioned into sets O_U and O_L that can be produced at U and L respectively[3]. Inputs and outputs will often be labelled in a manner that makes their port clear. For example, $?i_U$ is an input at U and $!o_L$ is an output at L. A *global tester* observes both ports and so observes a trace in $\mathcal{A}ct^*$, called a *global trace*. These traces can be transformed into two *local traces*.

Definition 4. *Let $\sigma \in \mathcal{A}ct^*$ and $p \in \mathcal{P}$. We let $\pi_p(\sigma)$ denote the projection of σ onto p; this is called a* local trace. *The function π_p can be defined by the following rules.*

1. $\pi_p(\epsilon) = \epsilon$.
2. *If* $z \in (I_p \cup O_p \cup \{\delta\})$ *then* $\pi_p(z\sigma) = z\pi_p(\sigma)$.
3. *If* $z \in I_q \cup O_q$, *for* $q \neq p$, *then* $\pi_p(z\sigma) = \pi_p(\sigma)$.

Given global traces $\sigma, \sigma' \in \mathcal{A}ct^*$ we write $\sigma \sim \sigma'$ if σ and σ' cannot be distinguished in the distributed test architecture. Thus, $\sigma \sim \sigma'$ if and only if $\pi_U(\sigma) = \pi_U(\sigma')$ and $\pi_L(\sigma) = \pi_L(\sigma')$.

The **dioco** implementation relation has been devised for testing in the distributed test architecture using a function *in* that returns the input potion of a trace [20].

Definition 5. *Let $s, i \in \mathcal{IOTS}(I, O)$. We write i* **dioco** *s if for every trace σ such that $i \stackrel{\sigma}{\Longrightarrow} i'$ for some i' that is in a quiescent state, if there is a trace $\sigma_1 \in \mathcal{T}r(s)$ such that $in(\sigma_1) \sim in(\sigma)$ then there exists a trace $\sigma' \in \mathcal{T}r(s)$ such that $s \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$.*

[3] An alternative, used in [20], is to allow outputs to be tuples of values but the formalism used in this paper has the advantage of simplifying the notation and analysis.

Only traces reaching quiescent states are considered in **dioco** since these allow us to put together the local traces; a non-quiescent state can receive additional output at a port and this situation is uncontrollable [20]. Since in this paper all processes are input-enabled we can simplify the previous definition.

Proposition 1. *Given $s, i \in \mathcal{IOTS}(I,O)$, if s and i are input-enabled then we have that i* **dioco** *s if and only if for every trace σ such that $i \stackrel{\sigma}{\Longrightarrow} i'$ for some i' that is in a quiescent state, there exists a trace σ' such that $s \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$.*

3 Test cases for the distributed test architecture

A test case is a process with a finite number of states that interacts with the SUT. A test case may correspond to a test objective: It may be intended to examine some part of the behaviour of the SUT. When designing test cases it is thus simpler to consider *global test cases*, that is, test cases that can interact with all of the ports of the system. However, in the distributed test architecture we do not have a global tester that can apply a global test case: Instead we place a *local tester* at each port. The local tester at port p only observes the behaviour at p and can only send input to the SUT at p.

A global test case is an IOTS that has the same input and output sets as the specification process s. A *local test case* is a tuple containing a test case for each of the available ports and has the input and outputs sets corresponding to its port. If $s \in \mathcal{IOTS}(I,O)$ then every global test case for s is a process from $\mathcal{IOTS}(I, O \cup \{\delta\})$. In our setting, with two ports, a local test case for s is a pair (t_U, t_L) such that $t_p \in \mathcal{IOTS}(I_p, O_p \cup \{\delta\})$ $(p \in \{U, L\})$. Test cases, either global or local, synchronize on values with either specifications or SUTs and do not contain internal actions. As usual, (global or local) test cases cannot block output from the SUT: If the SUT produces an output then the test case should be able to record this situation. Thus, for every state t' of a test case t and output $!o \in O$ we have that $t' \stackrel{!o}{\longrightarrow}$. $\perp$ is the global test case that cannot send input to the SUT and thus whose traces are all elements of $(O \cup \{\delta\})^*$. We let $\perp_p$ denote the corresponding local tester for port p, whose set of traces is $(O_p \cup \{\delta\})^*$.

Definition 6. *Let $s \in \mathcal{IOTS}(I,O)$ be an IOTS with port set $\mathcal{P} = \{U, L\}$. A* local test case *is a tuple (t_U, t_L) of* local testers *in which for all $p \in \mathcal{P}$ we have that t_p is a test case in $\mathcal{IOTS}(I_p, O_p \cup \{\delta\})$. In addition, if $t_p \stackrel{\sigma}{\Longrightarrow} t'_p$ for some σ then $t'_p \stackrel{!o_p}{\Longrightarrow}$ for all $!o_p \in O_p \cup \{\delta\}$.*

The following function, defined in [20], takes a global test case and returns local testers.

Definition 7. *Given global test case t and port p, $local_p(t)$ denotes the local tester at p defined by the following rules.*

1. *If t is the null process $\perp$ then $local_p(t)$ is $\perp_p$.*

2. *If* $a \in I_p \cup O_p \cup \{\delta\}$ *then* $local_p(a.t) = a.local_p(t)$.
3. *If* $a \in I_q \cup O_q$, $q \neq p$, *then* $local_p(a.t) = local_p(t)$.
4. $local_p(t_1 + \cdots + t_k) = local_p(t_1) + \cdots + local_p(t_k)$.

Next we introduce a notion of parallel composition between a system and a (global or local) test case.

Definition 8. *Let* $s \in \mathcal{IOTS}(I, O)$, t *be a global test case for* s *and* $t' = (t_U, t_L)$ *be a local test case for* s. *We introduce the following notation.*

1. $s||t$ *denotes the application of* t *to* s. *The system* $s||t$ *belongs to* $\mathcal{IOTS}(I, O)$ *and is formed by* s *and* t *synchronizing on all visible actions. Internal actions can be autonomously performed by* s.
2. $s||t'$ *denotes the application of* t' *to* s, *often represented by* $s||t_U||t_L$. $s||t'$ *belongs to* $\mathcal{IOTS}(I, O)$ *and it is formed from* s *and* t' *by* s *and* t_U *synchronizing on actions in* $I_U \cup O_U$ *and by* s *and* t_L *synchronizing on actions in* $I_L \cup O_L$. s, t_U, *and* t_L *synchronize on* δ. *Internal actions can be autonomously performed by* s.
3. *Since* $s||t$ *and* $s||t'$ *are systems, the notation introduced in Definition 2 can be applied to them.*
4. *We let* $\mathcal{T}r(s, t)$ *(resp.* $\mathcal{T}r(s, t')$*) denote the set of traces that can result from* $s||t$ *(resp.* $s||t'$*) and their prefixes.*

The following notation is used in order to reason about the application of test cases to systems.

Definition 9. *Let* $s, i \in \mathcal{IOTS}(I, O)$ *and* t *be a test case.*

1. *A trace* σ *is a* test run *for* i *with* t *if* $i||t \stackrel{\sigma\delta}{\Longrightarrow}$ *(and so at the end of this test run the SUT is quiescent).*
2. *Implementation* i **passes** *test run* σ *with* t *for* s *if there exists* $\sigma' \in \mathcal{T}r(s)$ *such that* $\sigma' \sim \sigma$. *Otherwise* i **fails** σ *with* t *for* s.
3. *Implementation* i **passes** *test case* t *for* s *if* i **passes** *every possible test run of* i *with* t *for* s *and otherwise* i **fails** t.

A local test case t is said to be *deterministic* for a specification s if the interaction between s and t cannot reach a situation in which both local testers are capable of sending input [20] since in such a situation, the order in which these inputs are received by the SUT cannot be known.

Definition 10. *Let* $s \in \mathcal{IOTS}(I, O)$. *We say that the local test case* (t_U, t_L) *is* deterministic for s *if there do not exist traces* σ_1 *and* σ_2, *with* $\sigma_2 \sim \sigma_1$, *and* $a_1, a_2 \in I$, *with* $a_1 \neq a_2$, *such that* $s||t_U||t_L \stackrel{\sigma_1 a_1}{\Longrightarrow}$ *and* $s||t_U||t_L \stackrel{\sigma_2 a_2}{\Longrightarrow}$.

The local testers being deterministic does not guarantee that the local test case is deterministic. For example, deterministic local testers t_U and t_L could both start by sending input to the SUT.

Now let us consider a specification s such that $\mathcal{T}r(s)$ is the set of prefixes of $?i_U!o_L!o_U?i_L$ plus the traces obtained by completing this to make it input-enabled. We could have a local test case (t_U, t_L) in which t_U sends $?i_U$ and expects to observe $!o_U$ and t_L sends $?i_L$ after observing $!o_L$. Then (t_U, t_L) is deterministic for s but t_L does not know when to send $?i_L$ and this is a form of nondeterminism. We obtain the same problem with the corresponding global test case if we wish to apply it in the distributed test architecture. The following, which adapts a definition from [22] for nondeterministic finite state machines, is a necessary and sufficient condition under which we avoid this form of nondeterminism.

Definition 11. *A (local or global) test case t is* controllable *for IOTS s if there does not exist port $p \in \mathcal{P}$, $\sigma_1, \sigma_2 \in \mathcal{T}r(s,t)$ and $?i_p \in I_p$ with $\sigma_1?i_p \in \mathcal{T}r(s,t)$, $\sigma_2?i_p \notin \mathcal{T}r(s,t)$ and $\pi_p(\sigma_1) = \pi_p(\sigma_2)$.*

If a test case is controllable then, as long as no failures occur in testing, each input is supplied by a local tester at the point specified in the test case.

Proposition 2. *Let us suppose that we are testing $i \in \mathcal{IOTS}(I,O)$ with a test case t that is controllable for specification $s \in \mathcal{IOTS}(I,O)$. If an input $?i$ is supplied after $\sigma \in \mathcal{T}r(s,t)$ then $\sigma?i \in \mathcal{T}r(t)$.*

Proof. Proof by contradiction: assume that $?i$ is supplied after the trace $\sigma \in \mathcal{T}r(s,t)$, where $\sigma?i \notin \mathcal{T}r(t)$. Since $?i$ is supplied after σ at a port $p \in \mathcal{P}$ there exists a trace $\sigma'?i \in \mathcal{T}r(s,t)$ such that $\pi_p(\sigma) = \pi_p(\sigma')$. Thus, there exists $\sigma, \sigma' \in \mathcal{T}r(s,t)$, port $p \in \mathcal{P}$ and input $?i \in I_p$ that should be sent after σ' but not after σ and $\pi_p(\sigma) = \pi_p(\sigma')$. This contradicts t being controllable for s as required. □

If a test case t is controllable for s and a global trace $\sigma \in \mathcal{T}r(s,t)$ has been produced then each local tester knows what to do next (apply an input or wait for output). It is natural to now ask whether we can always implement a controllable global test cases using a controllable local test case.

Proposition 3. *If t is a global test case for $s \in \mathcal{IOTS}(I,O)$ and $t_p = local_p(t)$ for $p \in \{U, L\}$ then:*

1. *$\mathcal{T}r(s,t) \subseteq \mathcal{T}r(s,(t_U,t_L))$*
2. *$\mathcal{T}r(s,(t_U,t_L)) \subseteq \mathcal{T}r(s,t)$ if and only if t is controllable.*

Proof. It is straightforward to prove that for all $\sigma \in \mathcal{T}r(t)$ and $p \in \mathcal{P}$ there exists $\sigma_p \in \mathcal{T}r(local_p(t))$ such that $\sigma_p = \pi_p(\sigma)$. The first part of the result follows from the fact that $\mathcal{T}r(s,t) = \mathcal{T}r(s) \cap \mathcal{T}r(t)$ and $\mathcal{T}r(s,(t_U,t_L)) = \mathcal{T}r(s) \cap \mathcal{T}r((t_U,t_L))$, where $\mathcal{T}r((t_U,t_L))$ is the set of traces formed from interleavings of traces in $\mathcal{T}r(t_U)$ and $\mathcal{T}r(t_L)$ and so $\mathcal{T}r(t) \subseteq \mathcal{T}r((t_U,t_L))$.

Now assume that t is controllable and we prove that for all $\sigma \in \mathcal{T}r(s,(t_U,t_L))$ we have that $\sigma \in \mathcal{T}r(s,t)$, using proof by induction on the length of σ. Clearly the result holds for the base case $\sigma = \epsilon$. Thus, let us assume that the result holds for all traces of length less than $k > 0$ and σ has length k. Thus, $\sigma = a\sigma'$ for some $a \in \mathcal{A}ct$.

If $a = \delta$ then $t_U \xrightarrow{a} t'_U$, $t_L \xrightarrow{a} t'_L$, and $t \xrightarrow{a} t'$ for some t'_U, t'_L, t' and the result follows by observing that $t'_U = local_U(t')$, $t'_L = local_L(t')$, and t' is controllable for the process s' such that $s \stackrel{a}{\Longrightarrow} s'$ and thus by applying the inductive hypothesis to σ'.

Let us assume that $a \in I \cup O$. Without loss of generality, a occurs at U and so $a \in I_U \cup O_U$. Thus, there exists t'_U such that $t_U \xrightarrow{a} t'_U$. Since $t_U = local_U(t)$ it must be possible to have event a at U in t before any other event at U and so there must exist some minimal $\sigma_L \in (I_L \cup O_L)^*$ such that $\sigma_L a \in \mathcal{T}r(t)$. But $\pi_U(\sigma_L) = \pi_U(\epsilon)$ and so, since t is controllable for s, by the minimality of σ_L we must have that $\sigma_L = \epsilon$. Thus, there exists t' such that $t \xrightarrow{a} t'$. The result now follows observing that $t'_U = local_U(t')$, $t_L = local_L(t')$, t' is controllable for the process s' such that $s \stackrel{a}{\Longrightarrow} s'$ and by applying the inductive hypothesis to σ'.

Next we prove the left to right implication: We assume that $\mathcal{T}r(s, (t_U, t_L)) \subseteq \mathcal{T}r(s, t)$ and will prove that t is controllable for s. The proof is by contradiction. We assume that t is not controllable for s and so there exist $\sigma_1, \sigma_2 \in \mathcal{T}r(s, t)$ and port $p \in \mathcal{P}$ such that $\pi_p(\sigma_1) = \pi_p(\sigma_2)$ and there exists $?i_p \in I_p$ such that $\sigma_1 ?i_p \in \mathcal{T}r(s, t)$ and $\sigma_2 ?i_p \notin \mathcal{T}r(s, t)$. But clearly, since $\sigma_1 ?i_p \in \mathcal{T}r(s, t)$, we have that $\pi_p(\sigma_1)?i_p \in \mathcal{T}r(t_p)$ and so, since $\pi_p(\sigma_1) = \pi_p(\sigma_2)$ we have that $\pi_p(\sigma_2)?i_p \in \mathcal{T}r(t_p)$. Further, for $q \in \mathcal{P}$, $q \neq p$, we have that $\pi_q(\sigma_2) \in \mathcal{T}r(t_q)$ and so $\sigma_2 ?i_p \in \mathcal{T}r((t_U, t_L))$. Finally, since $\sigma_2 \in \mathcal{T}r(s, t)$ and s is input enabled we have that $\sigma_2 ?i_p \in \mathcal{T}r(s)$. Thus, we conclude that $\sigma_2 ?i_p \in \mathcal{T}r(s, (t_U, t_L))$ as required. □

The controllability of a test case t for s is defined in terms of the traces that can be produced by $t||s$. Thus, if two test cases define the same sets of traces when applied to s then either both are controllable or neither is controllable. The proof of the following is immediate from Proposition 3 and Definition 11.

Proposition 4. *Let t be a global test case for $s \in \mathcal{IOTS}(I, O)$ and $t' = (local_U(t), local_L(t))$. Then t' is controllable for s if and only if t is controllable for s.*

Proposition 5. *Let t be a global test case and $t_p = local_p(t)$, $p \in \mathcal{P}$. If t is controllable for s then (t_U, t_L) is deterministic for s. However, it is possible for t to be deterministic for s but for t not to be controllable for s.*

Proof. We have seen that a test case can be deterministic for s and not controllable for s. We therefore assume that t is controllable for s and use proof by contradiction and so assume that (t_U, t_L) is not deterministic for s. By Proposition 3, $\mathcal{T}r(s, t) = \mathcal{T}r(s, (t_U, t_L))$. Since (t_U, t_L) is not deterministic for s there exist $\sigma_1, \sigma_2 \in \mathcal{T}r(s, t)$ with $\sigma_1 \sim \sigma_2$ and inputs $?i_U \in I_U$ and $?i_L \in I_L$ such that $\sigma_1 ?i_U, \sigma_2 ?i_L \in \mathcal{T}r(s, t)$. Since global test cases are deterministic, in t it is possible to input $?i_U$ after σ_1 but not after σ_2, both of these are in $\mathcal{T}r(s, t)$ and $\pi_U(\sigma_1) = \pi_U(\sigma_2)$. Thus, t is not controllable as required. □

4 Deciding whether a test case is controllable

It is clear that it is desirable to apply controllable test cases since this allows each local tester to know when to apply an input. In this section we show that

it is possible to determine whether a local test case is controllable in low order polynomial time. This result will be used in Section 5 in a test generation algorithm that returns controllable test cases.

The work reported in [23, 24] investigated Message Sequence Charts (MSCs) and whether a set of MSCs implies other MSCs[4]. This is achieved by considering the language defined by a set of MSCs. A trace can be seen as an MSC and in this section we use a definition and a complexity result from [23].

Let us consider an MSC with message alphabet Σ. If message $a \in \Sigma$ is sent from process p to process q then the sending of a is represented by event $send(p, q, a)$ and the receiving of a is represented by event $receive(p, q, a)$. $\hat{\Sigma}$ denotes the set of send and receive events, $\hat{\Sigma}^S$ denotes the set of send events and $\hat{\Sigma}^R$ denotes the set of receive events. An MSC with processes $P_1, \ldots, P_k$ and alphabet Σ can be defined by the following:

1. A set E of events, partitioned into a set S of send events and a set R of receive events.
2. A mapping *occ* such that each event in E is mapped to the process on which it occurs.
3. A bijective function f between send and receive events.
4. A mapping *label* from events to elements of $\hat{\Sigma}$. Naturally, for $e \in E$ we must have that $label(e) \in \hat{\Sigma}^S$ if and only if $e \in S$.
5. For each process P_i a total ordering $\leq_i$ on the events that occur at P_i such that the transitive closure of the relation

$$\bigcup_i \leq_i \ \cup \{(s, r) | s \in S, r \in R, r = f(s)\}$$

is a partial order on E. We include $\{(s, r) | s \in S, r \in R, r = f(s)\}$ since a message cannot be received before it is sent.

An MSC defines a partial order on E and a language with alphabet $\hat{\Sigma}$ that preserves this partial order: The set of words that are linearizations of the MSC. A word is *well-formed* if for every receive event there is a corresponding earlier send event. [23] looks at projections of words onto the processes of an MSC. In our case, each local tester is a process and the SUT is also a process. Given word w and process p, $w|_p$ is the sequence of events in w at p. We will use the following closure condition [23].

Definition 12. *A language L over $\hat{\Sigma}$ satisfies closure condition CC3 if and only if for every well-formed word w we have that if for each process p there is a word $v^p \in pre(L)$ such that $w|_p = v^p|_p$, then w is in $pre(L)$.*

A (global or local) test case defines a set of traces and each trace defines an MSC. Results in [23] concern asynchronous communications but they apply when communications and synchronous [24] by restricting the language defined by a set of MSCs to only include words that are consistent with synchronous

[4] They restrict attention to basic MSCs, which we call MSCs.

communications. Given a (global or local) test case t and specification s, $L(s,t)$ will denote the language of words that are defined by the MSCs corresponding to traces in $\mathcal{T}r(s,t)$ where communications are synchronous and so every send is immediately followed by the corresponding receive. We now prove that test case t is controllable for s if and only if the corresponding language satisfies CC3.

Proposition 6. *Given $s \in \mathcal{IOTS}(I,O)$, test case t is controllable for s if and only if the language $L = L(s,t)$ satisfies closure condition CC3.*

Proof. First, assume that t is not controllable for s and we will prove that L does not satisfy CC3. Since t is not controllable for s there exist a port $p \in \mathcal{P}$, traces $\sigma_1, \sigma_2 \in \mathcal{T}r(s,t)$, and input $?i_p$ at p such that $\pi_p(\sigma_1) = \pi_p(\sigma_2)$, $\sigma_1 ?i_p \in \mathcal{T}r(t)$ but $\sigma_2 ?i_p \notin \mathcal{T}r(t)$.

Let us consider $w \in \hat{\Sigma}$ that corresponds to σ_2 followed by $send(p, SUT, ?i_p)$. For every $q \in \mathcal{P} \cup \{SUT\}$ such that $q \neq p$ we have that $w|_q$ is a projection of the string from $pre(L)$ corresponding to σ_2 and so there is some $v^q \in pre(L)$ such that $w|_q = v^q|_q$. Since $\pi_p(\sigma_1) = \pi_p(\sigma_2)$ we know that there is some $v^p \in pre(L)$ such that $w|_p = v^p|_p$. Thus, since $w \notin L$, L does not satisfy CC3 as required.

Now assume that L does not satisfy CC3 and we will prove that t is not controllable for s. Since L does not satisfy CC3 there is a well-formed word $w \notin pre(L)$ such that for all $p \in \mathcal{P} \cup \{SUT\}$ there is some $v^p \in pre(L)$ such that $w|_p = v^p|_p$. Let w be a shortest such word and $w = w'e$ for some event e. Clearly w' is well-formed and for all $p \in \mathcal{P} \cup \{SUT\}$ there is some $v^p \in pre(L)$ such that $w'|_p = v^p|_p$. By the minimality of w, $w' \in pre(L)$.

If e is a receive event then there is a corresponding send event e' that is an event for some process in w. Thus, since $w' \in pre(L)$ and communications are synchronous we must have that $w'e \in pre(L)$, providing a contradiction, and so e must be a send event. Now let us suppose that e is the sending of a message from process SUT. Then since $w'e|_{SUT} = v^{SUT}|_{SUT}$ for some $v^{SUT} \in pre(L)$, after $w'|_{SUT}$ the SUT must be able to send e and so $w = w'e \in pre(L)$, providing a contradiction. Thus e is the sending of an input $?i_p$ from the local tester at some port p to the SUT.

Let σ_2 denote the sequence from $\mathcal{T}r(s,t)$ that corresponds to w': There must be some such word since w' is well-formed and can be followed by a send event and communications are synchronous. There is a shortest $v^p \in pre(L)$ such that $w'e|_p = v^p|_p$. Let $v_1^p \in pre(L)$ be equal to the sequence v^p with the event e removed. Let σ_1 denote the sequence from $\mathcal{T}r(s,t)$ that corresponds to v_1^p. It is possible to follow σ_1 by the sending of $?i_p$ in $\mathcal{T}r(s,t)$ but it is not possible to follow σ_2 by the sending of $?i_p$ in $\mathcal{T}r(s,t)$ and $\pi_p(\sigma_1) = \pi_p(\sigma_2)$. Thus, t is not controllable for s as required. □

Given k MSCs with n processes, if r is the total number of events in the MSCs then it is possible to decide whether the language defined by the MSCs satisfies CC3 in time of $O(k^2 n + rn)$ [23]. Thus, this result holds if $\mathcal{T}r(s,t)$ contains k traces and in our case $n = 3$. In addition, if l is an upper bound on the lengths of traces in $\mathcal{T}r(s,t)$ then there are at most $r = 2lk$ events. Thus, the worst time complexity is of $O(k^2 + lk)$.

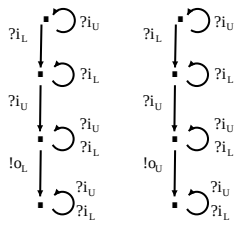

Fig. 2. Processes s_1 and i_1.

5 An implementation relation for controllable testing

We obtain a new implementation relation if we restrict testing to the use of controllable test cases.

Definition 13. *Given $s, i \in \mathcal{IOTS}(I, O)$ we write i* **c-dioco** *s if for every controllable local test case (t_U, t_L) we have that i* **passes** *(t_U, t_L) for s.*

We now investigate how **c-dioco** relates to **dioco** and **ioco**; **dioco** is strictly weaker than **ioco** if all processes are input enabled [20].

Proposition 7. *If i* **dioco** *s then i* **c-dioco** *s. Further, there exists processes i_1 and s_1 such that i_1* **c-dioco** *s_1 but we do not have that i_1* **dioco** *s_1.*

Proof. The first part follows from **c-dioco** restricting consideration to controllable local test cases. Consider the processes s_1 and i_1 shown in Figure 2, which are incomparable under **dioco**. The only controllable local test cases involve input at no more than one port and for each such test case neither process can produce output. We therefore have that i_1 **c-dioco** s_1 as required. □

We now define an algorithm that returns a *complete* test suite. We talk of *completeness in the limit*, requiring that for all non-conforming SUT the algorithm will eventually produce a controllable local test case that will be failed by the SUT. Completeness is usually split into two results. A test suite is *sound* if conforming implementations pass all the tests in the suite. It is *exhaustive* if non-conforming implementations fail some of the tests. Our algorithm, given in Figure 3, is an adaption of an algorithm to derive test cases from timed FSMs [19]. This algorithm is *non-deterministic* since in each recursive step it can proceed in different ways. Each election generates a different controllable local test case. By applying the algorithm in all the possible ways we generate a test suite from a specification s that we call $tests(s)$.

The algorithm keeps track of states of the local testers by storing tuples (Q', s^U, s^L) that indicate that the specification could be in any of the states in Q'. We construct test cases by iteratively applying one of six possibilities. The first two return a minimal local tester and are the base cases. The third and

fourth consider the possibility of adding an input. First, it is necessary to check that the addition of this input will produce a controllable local tester. If this is the case, we add a transition labelled by this input to the corresponding local tester, updating the set of auxiliary tuples and considering all possible outputs at the corresponding port. If the output is expected by the specification then we can continue testing after receiving the output; otherwise, we should reach a state where we do not provide further input. The fifth and sixth consider the case where a local tester patiently waits to receive an output; we also have to consider the possibility of receiving no output represented by δ.

Now we show the completeness of $tests(s)$. First, using Definition 13, it is obvious that i **c-dioco** s implies that for every local test case $t \in tests(s)$ we have i **passes** t for s. The other implication is shown in the following result.

Proposition 8. *Let $s, i \in \mathcal{IOTS}(I, O)$. If for every local test case $t \in tests(s)$ we have i **passes** t for s then we also have i **c-dioco** s.*

Proof. We use proof by contradiction: Assume that i **c-dioco** s does not hold and we have to prove that i fails a controllable test case belonging to $tests(s)$.

Since i **c-dioco** s does not hold, there exists a controllable test case t_1 such that i **fails** t_1 for s. If t_1 belongs to $tests(s)$ then we conclude the proof. Otherwise, i must fail a test run with t_1. Consider a minimal length sequence $\sigma = a_1 a_2 \dots a_n \in \mathcal{A}ct^*$ such that i **fails** σ for t_1. Due to the minimality of σ, for every proper prefix σ_1 of σ such that i can be quiescent after σ_1, there exists $\sigma_1' \in \mathcal{T}r(s)$ such that $\sigma_1' \sim \sigma_1$. The following algorithm defines a controllable local test case (t_U, t_L), $t_p = (Q_p, I_p, O_p, T_p, q_{in}^p)$ for $p \in \{U, L\}$, that belongs to $tests(s)$ and is failed by i.

$S_{aux} := \{(q_{in}$ **after** $\epsilon, q_{in}^U, q_{in}^L)\}$; $Q_U := \{q_{in}^U\}$; $Q_L := \{q_{in}^L\}$; $T_U, T_L := \emptyset$;
$aux := (q_{in}$ **after** $\epsilon, q_{in}^U, q_{in}^L)$; {`We have` $aux := (Q_{aux}, s_{aux}^U, s_{aux}^L)$}
for $j := 1$ to n do
  Consider aux {aux `belongs to` S_{aux}};
  if $a_j \in I_U$ then Apply case 3 of Algorithm; $aux := (Q_{aux}$ **after** $a_j, q', s_{aux}^L)$;
  if $a_j \in I_L$ then Apply case 4 of Algorithm; $aux := (Q_{aux}$ **after** $a_j, s_{aux}^U, q')$;
  if $a_j \in O_U$ then Apply case 5 of Algorithm; $aux := (Q_{aux}$ **after** $a_j, q', s_{aux}^L)$;
  if $a_j \in O_L$ then Apply case 6 of Algorithm; $aux := (Q_{aux}$ **after** $a_j, s_{aux}^U, q')$;
{`Now we apply the base cases`}
for all non-initialized s_U such that $\exists Q', s_L : (Q', s_U, s_L) \in S_{aux}$ do $s_U := \perp_U$;
for all non-initialized s_L such that $\exists Q', s_U : (Q', s_U, s_L) \in S_{aux}$ do $s_L := \perp_L$

It is clear that this local test case belongs to $tests(s)$, the last two steps corresponding to the application of the base cases. Moreover, $i||t_U||t_L \stackrel{\sigma}{\Longrightarrow}$ since, by construction, σ is a trace of (t_U, t_L). Thus, since there does not exist $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \sim \sigma$, we obtain i **fails** (t_U, t_L), concluding i **passes** (t_U, t_L) for s does not hold. □

From Proposition 8 we immediately obtain the desired result.

Corollary 1. *Let $s, i \in \mathcal{IOTS}(I, O)$. We have i* **c-dioco** *s if and only if for every local test case $(t_U, t_L) \in tests(s)$ we have that i* **passes** *(t_U, t_L) for s.*

6 Conclusions

If we are testing a state based system with physically distributed interfaces/ports then we place a tester at each port. If these testers cannot communicate with one another and there is no global clock then we are testing in the distributed test architecture. While testing in the distributed test architecture has received much attention, most previous work considered the problem of testing against a deterministic finite state machine. Only recently has the problem of testing from an input-output transition system (IOTS) been investigated.

This paper considered what it means for a test case to be controllable in the distributed test architecture, showing that it is not sufficient for a test case to be deterministic. A new implementation relation, **c-dioco**, was defined: This corresponds to controllable testing from an IOTS in the distributed test architecture.

We have shown that it is possible to decide in low order polynomial time whether a test case is controllable. This allowed us to define a test generation algorithm. The algorithm is guaranteed to return controllable test cases and, in addition, any controllable test case can be returned by the algorithm. Thus, in the limit the test generation algorithm is complete.

There are several possible areas of future work. The test generation algorithm does not aim to return test cases that, for example, achieve a given test objective. It would therefore be interesting to combine this with approaches that direct test generation. It would also be interesting to investigate formalisms in which a transition can be triggered by the SUT receiving input at several ports [25].

References

1. ISO/IEC JTC 1, J.T.C.: International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts. ISO/IEC (1994)
2. Sarikaya, B., v. Bochmann, G.: Synchronization and specification issues in protocol testing. IEEE Transactions on Communications **32** (1984) 389–395
3. Luo, G., Dssouli, R., v. Bochmann, G.: Generating synchronizable test sequences based on finite state machine with distributed ports. In: 6th IFIP Workshop on Protocol Test Systems, IWPTS'93, North-Holland (1993) 139–153
4. Tai, K.C., Young, Y.C.: Synchronizable test sequences of finite state machines. Computer Networks and ISDN Systems **30**(12) (1998) 1111–1134
5. Rafiq, O., Cacciari, L.: Coordination algorithm for distributed testing. The Journal of Supercomputing **24**(2) (2003) 203–211
6. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines: A survey. Proceedings of the IEEE **84**(8) (1996) 1090–1123
7. Brinksma, E., Tretmans, J.: Testing transition systems: An annotated bibliography. In: 4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067, Springer (2001) 187–195

8. Petrenko, A., Yevtushenko, N.: Testing from partial deterministic FSM specifications. IEEE Transactions on Computers **54**(9) (2005) 1154–1165
9. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software – Concepts and Tools **17**(3) (1996) 103–120
10. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann (2007)
11. Hierons, R., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H.: Using formal methods to support testing. ACM Computing Surveys (in press) (2008)
12. Brinksma, E., Heerink, L., Tretmans, J.: Factorized test generation for multi-input/output transition systems. In: 11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98, Kluwer Academic Publishers (1998) 67–82
13. Núñez, M., Rodríguez, I.: Towards testing stochastic timed systems. In: 23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767, Springer (2003) 335–350
14. Brandán Briones, L., Brinksma, E.: A test generation framework for quiescent real-time systems. In: 4th Int. Workshop on Formal Approaches to Testing of Software, FATES'04, LNCS 3395, Springer (2004) 64–78
15. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: 11th Int. SPIN Workshop on Model Checking of Software, SPIN'04, LNCS 2989, Springer (2004) 109–126
16. Bijl, M.v., Rensink, A., Tretmans, J.: Action refinement in conformance testing. In: 17th Int. Conf. on Testing of Communicating Systems, TestCom'05, LNCS 3502, Springer (2005) 81–96
17. López, N., Núñez, M., Rodríguez, I.: Specification, testing and implementation relations for symbolic-probabilistic systems. Theoretical Computer Science **353**(1–3) (2006) 228–248
18. Frantzen, L., Tretmans, J., Willemse, T.: A symbolic framework for model-based testing. In: 1st Combined Int. Workshops on Formal Approaches to Software Testing and Runtime Verification, FATES 2006 and RV 2006, LNCS 4262, Springer (2006) 40–54
19. Merayo, M., Núñez, M., Rodríguez, I.: Formal testing from timed finite state machines. Computer Networks **52**(2) (2008) 432–460
20. Hierons, R., Merayo, M., Núñez, M.: Implementation relations for the distributed test architecture. In: Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047, Springer (2008) 200–215
21. Tretmans, J.: Testing concurrent systems: A formal approach. In: 10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664, Springer (1999) 46–65
22. Hierons, R.: Controllable testing from nondeterministic finite state machines with multiple ports (2008) Submitted.
23. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Transactions on Software Engineering **29**(7) (2003) 623–633
24. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. Technical report, University of Edinburgh (2003) Available at `http://homepages.inf.ed.ac.uk/kousha/msc_inference_j_v.ps`.
25. Haar, S., Jard, C., Jourdan, G.V.: Testing input/output partial order automata. In: Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581, Springer (2007) 171–185

Input: Specification $s = (Q, I, O, T, q_{in})$.
Output: Controllable local test case (t_U, t_L); $t_p = (Q_p, I_p, O_p, T_p, q^p_{in})$ for $p \in \{U, L\}$.

{`Initialization`}
$S_{aux} := \{(q_{in}$ **after** $\epsilon, q^U_{in}, q^L_{in})\}$; $Q_U := \{q^U_{in}\}$; $Q_L := \{q^L_{in}\}$; $T_U, T_L := \emptyset$;
while $S_{aux} \neq \emptyset$ do

- Choose $(Q', s^U, s^L) \in S_{aux}$;
- Choose one of the following six possibilities:
  1. {$\mathtt{s}^U$ `will be` $\perp_U$ `if it has not been defined yet`}
     (a) if $s^U \neq \perp_U$ then $s^U := \perp_U$;
     (b) If $s^L = \perp_L$ then $S_{aux} := S_{aux} - \{(Q', s^U, s^L)\}$;
     {`Remove the tuple if both local testers are` $\perp$}
  2. {$\mathtt{s}^L$ `will be` $\perp_L$ `if it has not been defined yet`}
     {`Similar to the previous case, substituting` U `by` L}
  3. {`Add an input at port` U `if the result is controllable`}
     (a) Let $?i_U \in I_U$ be such that $(\hat{t}_U, t_L)$ is controllable for s, where $\hat{t}_U$ is formed from t_U after making the following changes:
        i. $\hat{t}_U := (\hat{Q}_U, I_U, O_U, \hat{T}_U, \hat{q}^U_{in})$, where $\hat{Q}_U := Q_U$; $\hat{T}_U := T_U$;
        ii. $\hat{s}^U$ and $\hat{q}^U_{in}$ are the copies of s^U and q^U_{in} in $\hat{Q}_U$;
        iii. Consider a fresh state $\hat{q}' \notin \hat{Q}_U$; $\hat{Q}_U := \hat{Q}_U \cup \{\hat{q}'\}$; $\hat{q}' := \perp_U$;
        iv. For all $a \in O_U \cup \{\delta\} \cup \{?i_U\}$ do $\hat{T}_U := \hat{T}_U \cup \{(\hat{s}^U, a, \hat{q}')\}$;
        v. For all $\hat{s} \in \hat{Q}_U \setminus \{\hat{s}^U\}$ such that $\exists Q'', \hat{s}' : (Q'', \hat{s}, \hat{s}') \in S_{aux}$ do $\hat{s} := \perp_U$;
        vi. For all $\hat{s}$ such that $\exists Q'', \hat{s}' : (Q'', \hat{s}', \hat{s}) \in S_{aux}$ do $\hat{s} := \perp_L$
     (b) $S_{aux} := S_{aux} - \{(Q', s^U, s^L)\}$;
     (c) Consider a fresh state $q' \notin Q_U$; $Q_U := Q_U \cup \{q'\}$;
     (d) $T_U := T_U \cup \{(s^U, ?i_U, q')\}$; $S_{aux} := S_{aux} \cup \{(Q'$ **after** $?i_U, q', s^L)\}$;
     (e) For all $!o_U \in O_U$ such that $!o_U \notin out(Q')$ do
     {`These are unexpected outputs: Construct` $\perp_U$ `after them`}
        i. Consider a fresh state $q' \notin Q_U$; $Q_U := Q_U \cup \{q'\}$; $q' := \perp_U$;
        ii. $T_U := T_U \cup \{(s^U, !o_U, q')\}$
     (f) For all $!o_U \in O_U$ such that $!o_U \in out(Q')$ do
     {`These are expected outputs: Testing can continue after them`}
        i. Consider a fresh state $q' \notin Q_U$; $Q_U := Q_U \cup \{q'\}$;
        ii. $T_U := T_U \cup \{(s^U, !o_U, q')\}$; $S_{aux} := S_{aux} \cup \{(Q'$ **after** $!o_U, q', s^L)\}$
  4. {`Add an input at port` L `if the result is controllable`}
     {`Similar to the previous case, substituting` U `by` L}
  5. {`Wait for an output at port` U}
     (a) $S_{aux} := S_{aux} - \{(Q', s^U, s^L)\}$;
     (b) For all $o \in O_U \cup \{\delta\}$ such that $o \notin out(Q')$ do
     {`These are unexpected outputs: Construct` $\perp_U$ `after them`}
        i. Consider a fresh state $q' \notin Q_U$; $Q_U := Q_U \cup \{q'\}$; $q' := \perp_U$;
        ii. $T_U := T_U \cup \{(s^U, o, q')\}$;
     (c) For all $o \in O_U \cup \{\delta\}$ such that $o \in out(Q')$ do
     {`These are expected outputs: Testing can continue after them`}
        i. Consider a fresh state $q' \notin Q_U$ and let $Q_U := Q_U \cup \{q'\}$;
        ii. $T_U := T_U \cup \{(s^U, o, q')\}$; $S_{aux} := S_{aux} \cup \{(Q'$ **after** $o, q', s^L)\}$
  6. {`Wait for an output at port` L}
     {`Similar to the previous case, substituting` U `by` L}

Fig. 3. Controllable test cases generation algorithm.