ABSTRACT
We investigate the problem of specification based testing with dense sets of inputs and outputs, in particular with imprecision as they might occur due to errors in measurements, numerical instability or noisy channels. Using quantitative transition systems to describe implementations and specifications, we introduce implementation relations that capture a notion of correctness “up to $\varepsilon$”, allowing deviations of implementation from the specification of at most $\varepsilon$. These quantitative implementation relations are described as Hausdorff distances between certain sets of traces. They are conservative extensions of the well-known ioco relation. We develop an on-line and an off-line algorithm to generate test cases from a requirement specification, modeled as a quantitative transition system. Both algorithms are shown to be sound and complete with respect to the quantitative implementation relations introduced.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging;
D.2.8 [Software Engineering]: Metrics;

General Terms
Reliability, Theory.

Keywords

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT’08, October 19–24, 2008, Atlanta, Georgia, USA.
Copyright 2008 ACM 978-1-60558-468-3/08/10 ...$5.00.

1. INTRODUCTION
Testing is the most popular validation technique for software systems used in practice. At the same time, testing is expensive, taking from 40%-70% of all system development costs. Model-driven testing is an innovative technique that aims at reducing these costs by providing automated techniques for test case generation, execution and evaluation. Starting point is a formal model representing the system requirements specification, usually given as a transition system of some form. The first model-driven test theories [14, 6] considered the temporal order in which the events of the implementation-under-test (IUT) should take place. Recently, several extensions have been developed which surpass plain functional testing and also take into account quantitative information of the IUT: [3, 1, 10, 12] extend the classical model-driven test theories with real-time; [7, 8] with data, and [17] to hybrid systems. These papers provide a solid formal underpinning of real-time, hybrid and data testing, together with methods for automatic test case generation, execution and evaluation.

These theories, however, handle the numerical values contained within the requirement specification and the IUT with an infinite precision. That is, they do not take into account deviations from these values due to measurement errors, numerical instability or noisy channels: e.g., if the specification requires a response time of 1 second, but the IUT responds within 1.01 second, a fail verdict is generated, even though the deviation might be tolerable.

For real-time testing, [11] overcome this problem by explicitly modeling the tester’s time observation capabilities through a digital clock. Also in the area of verification, the realization that real-time models are idealized mathematical abstractions that may not be implementable in physical reality has led to different, more robust semantics for real-time models [9, 13, 4]. For systems where the numerical information represent different quantities than real-time, such as resources or physical phenomena, we are not aware of such theories, neither in testing nor in verification.

This paper presents a model-driven test theory in the presence of imprecisions: rather than concentrating on one particular area like timed or hybrid testing, we present a general theory for testing quantitative systems that works for systems containing numerical information, no matter how the numbers are interpreted. This allows us to focus on the essentials of testing with imprecise information; one can always specialize our theory to deal with the particularities of a concrete (real-time, hybrid, probabilistic) data domain.

We set our theory in the context of quantitative transi-
I approach, where test cases are first generated, and sub-mformed immediately. The second one is a batch or off-line (i.e. observing the IUT or providing some input) in per-an implementation and a specification through testing. The an distance even if unperturbed in transmission, might be interpreted as itself might also be a source of deviation, where an input, testing algorithms that estimate the at the tester, where again testing approaches are sound and complete with respect to \( \text{qico}_0 \), up to the perturbations of \( \varepsilon \). This means that if \( \varepsilon = \gamma + \delta \) and \( L(n) \)-distance \( S \), then no test case derived from \( S \) will report a distance of more than \( \varepsilon \) (soundness) and there exist a test case that gets arbitrary close to \( \varepsilon \) (completeness).

For space restrictions, the current paper does not contain proofs, for which we refer the reader to [2].

Structure of the paper.
In Section 2 we give a semi-formal introduction in the \( \text{ioco} \) theory. In Section 3 we introduce QTS. In Section 4 we intro-duce distances on sets of traces. In Section 5 we define the \( \text{qico} \), relations and analyze some of their properties. In Section 6 we introduce the on-the-fly testing algorithm for \( \text{qico} \) and prove its soundness and completeness. In Section 7 we introduce test cases. We conclude with Section 8.

2. IOCO TESTING
In this section we introduce the basic principles of the \( \text{ioco} \) testing theory [14], which are our starting point for the quantitative testing approach. Specification-based testing à la \( \text{ioco} \) is all about sequences of inputs and outputs, the so-called traces. The most interesting traces are the test execu-tions, which comprise inputs, as they are sent by the tester, and outputs, as they are returned by the implementation (c.f. Figure 1 (a)), in chronological order. It is assumed that the sets of inputs and outputs, \( L_1 \), \( L_0 \), are finite. A test execu-tion is thus formally a trace \( \sigma \in (L_1 \cup L_0)^* \), which is syn-thesized by tester and implementation as testing proceeds. The specification, which serves as input to the tester\(^2\), is a formal object which describes a set of traces \( T \subseteq (L_1 \cup L_0)^* \).

These specifications are usually labeled transition systems, specified by a process algebra or other formalisms. The correctness criterion which the tester employs is that every test execution \( \sigma \) must be element of \( T \). For space restrictions, the current paper does not contain proofs, for which we refer the reader to [2].

\( d_{\text{qico}} \) measures how far a system imple-mentation lies from a system specification. We present two testing algorithms that estimate the \( d_{\text{qico}} \)-distance between an implementation and a specification through testing. The first approach is on-line, and interleaves the test derivation and test execution phase. That is, each choice of the tester (i.e. observing the IUT or providing some input) in performed immediately. The second one is a batch or off-line approach, where test cases are first generated, and sub-sequently executed against the IUT. Both approaches are sound and complete with respect to \( \text{qico} \), up to the perturbations of \( \varepsilon \). This means that if \( \varepsilon = \gamma + \delta \) and \( L(n) \)-distance \( S \), then no test case derived from \( S \) will report a distance of more than \( \varepsilon \) (soundness) and there exist a test case that gets arbitrary close to \( \varepsilon \) (completeness).

For space restrictions, the current paper does not contain proofs, for which we refer the reader to [2].

\( \varepsilon \)We assume the tester to be a software tool.

\(^1\)We mark inputs with \?, outputs with !.

\(^2\)We assume the tester to be a software tool.
paradoxically denotes "lack of output observed". Therefore
the previous statement that $T \subseteq (L^1 \cup L^0)^*$ is inaccurate in the sense that the specification must also specify when quiescence is allowed to be observed. The specification must thus actually define a set of traces $T \subseteq (L^1 \cup L^0 \cup \{\delta\})^*$. ioco is a testing theory, which means that the whole approach informally explained above is actually described in a complete formal framework. Specifications and implementations are modeled as labeled transition systems defined on $L^1 \cup L^0$, and the correctness criterion is formalized as the implementation relation $ioco$, which gives the theory its name. On the formal level it is thus assumed that the set of traces of the implementation is also known. We assume that $I$ is this set. The implementation is $ioco$-conforming to the specification, if the following holds. For all $\sigma \in T$ and all $a' \in L^0$: $\sigma - a' \in I$ implies $\sigma - a' \in T$. The $ioco$ theory has a formal definition of test cases, which can be employed by the testing tool for offline testing. These test cases are usually described as deterministic, acyclic labeled transition systems defined over $(L^1 \cup L^0 \cup \{\delta\})$. It can be shown that the set of test cases is sound and complete. Soundness means that whenever a test case that is executed leads to a test failure, the tested implementation is indeed not $ioco$-conforming to the specification. Exhaustiveness means that, if all test cases are executed an appropriate number of times and report no failure, then the implementation is $ioco$-conforming. Exhaustiveness is of course only a theoretical result, since the set of test cases is usually infinite. Instead of executing pre-fabricated test cases it is also possible to conduct on-the-fly testing, i.e. to generate and execute test cases simultaneously. This is depicted in Algorithm 1, where procedure $ioco_{OTF}$ is called with parameters $I$ (the implementation), trace set $T$ as the specification, and the maximal number of test steps $n$. Local variable $\sigma$ denotes the current test execution and is initialized with $\lambda$ (the empty trace). The while-loop (line 3) is left either if $n$ test steps have been executed, or a test failure is observed (line 10). The tester either supplies an input to the implementation, if one exists (line 4), or receives and output (line 8). The choice is nondeterministic. In either case, $\sigma$ is updated with the new input or output (lines 6 and 9, respectively). If in line 10 the condition $\sigma \notin T$ is true, testing ends with a failure. If the while-loop is terminated, a pass verdict is returned. This algorithm is essentially implemented in the testing-tool TorX [15].

The concepts of implementation relation, on-the-fly testing, test cases and test executions will be extended for quantitative testing.

3. QTS

This section introduces quantitative transition systems (abbreviated QTS). These are labeled transition systems whose actions $a(x)$ consist of a label $a$ and a value $x \in [0,1]$. We start with some notation.

Let $A$ be any set. Then $A^*$ is the set of all finite sequences over $A$. We denote the concatenation of sequences $\sigma, \rho \in A^*$ by juxtaposition, i.e. as $\sigma \rho$.

For $\rho \in A^*$, we say that $\sigma$ is a prefix of $\rho$, if $\rho = \sigma \rho'$ for some $\sigma' \in A^*$. We say that $\sigma$ is a suffix of $\rho$, if $\rho = \sigma \rho'$ for some $\sigma' \in A^*$. If $\sigma$ is a prefix of $\rho$, we write $\sigma \preceq \rho$. We call $\sigma$ a proper prefix of $\rho$, denoted $\sigma < \rho$, if $\sigma \preceq \rho$, but $\sigma \neq \rho$. The empty sequence is denoted by $\lambda$. For a sequence $\sigma = a_1 a_2 \ldots a_n$, we write $|\sigma| = n$ for the length of $\sigma$; $|\sigma|_i = a_i$ for $i$'th symbol in $\sigma$; last($\sigma$) = $a_n$ for the last symbol in $\sigma$; and $\sigma^* = \sigma \sigma_{i+1} \ldots$ for the suffix of $\sigma$ starting at position $i$.

Definition 3.1 The tuple $Q = (S, S^0, L, \rightarrow)$ is a quantitative transition system iff (1) $S$ is a (possibly uncountable) set of states; (2) $S^0 \subseteq S$ is a set of initial states; (3) $L$ is a set of action labels, which is partitioned into two sets $(L^1, L^0)$ of input and output labels respectively. We write $A_i = L^1 \times [0,1], A_O = L^0 \times [0,1]$ and $A = L \times [0,1]$, for the sets of input, output and all actions. (4) $\rightarrow \subseteq S \times A \times S$ is the transition relation. For states $s, s' \in S$, $\alpha \in A$, we write $s \xrightarrow{\alpha} s'$ for $(s, \alpha, s') \in \rightarrow$ and $s \xrightarrow{a} s'$, if $\exists \alpha \in A_O: s \xrightarrow{\alpha} s'$. We denote by out($s$) = $\{\alpha \in A_O | s \xrightarrow{\alpha} \}$ the set of output actions that are enabled in $s$.

We denote the components of $Q$ by $S_Q, S^0_Q, L_Q, A_Q$, etc. and omit the subscripts when no confusion arises.

Actions $(a, x) \in A$ are denoted as $a(x)$; input labels and actions as $a?$ and $a?(x)$; and output labels and actions as $a!$ and $a!(x)$.

In order to make life easier, we assume that all considered QTS $(S, S^0, L, \rightarrow)$ be non-blocking on outputs, i.e. for all states $s \in S$, out($s$) $\neq \emptyset$. This relieves us from the duty to consider quiescence explicitly (c.f. Section 2), since the $\delta$-label can actually be treated as as output. This is no restriction. If the need arises to transform a QTS into a non-blocking one, we can extend $L^0$ with a label $\delta$, and add to every state $s \in S$ which is blocking on outputs (i.e. without any outgoing output-transition) a transition $s \xrightarrow{a\delta} s$.

This is analogous to constructing a suspension-automaton (c.f. [14]).

Definition 3.2 (Determinism) A QTS $Q$ is said to be deterministic if for for $s, s', s'' \in S, \alpha \in A$: $s \xrightarrow{\alpha} s'$ and $s' \xrightarrow{\alpha} s''$ implies $s' = s''$; $Q$ is input-enabled iff for all $s \in S, a? \in A_I$ we have $s \xrightarrow{a?}$.

Definition 3.3 (Traces) An execution fragment of $Q$ is a finite sequence $\nu = s_0 a_1 s_1 a_2 s_2 \ldots s_n$, such that $s_{i-1} \xrightarrow{a_i} s_i$, for all $1 \leq i \leq n$. The trace of $\nu$ is obtained by removing all states in $\nu$, i.e. trace($\nu$) = $a_1 \ldots a_n$. We then write $s_0 \xrightarrow{a_1 \ldots a_n} s_n$. We denote by $\text{tr}(Q) \subseteq A^*$ the set of all traces $\sigma$ of $Q$ starting in some starting state of $Q$.
4. METRICS FOR QTS

4.1 Distances and Hausdorff distances

Let \( X \) be a set. A distance on \( X \) is a function \( d : X \times X \to \mathbb{R}^+ \), such that \( d(x, x) = 0 \) and \( d(x, y) + d(y, z) \geq d(x, z) \) (triangle inequality).

We can lift any distance \( d \) on \( X \) to a distance on sets via the Hausdorff distance \( h^d : \mathcal{P}(X) \times \mathcal{P}(X) \to \mathbb{R}^+ \), which is defined as \( h^d(Y, Z) = \sup_{y \in Y} \inf_{z \in Z} d(y, z) \) for all \( Y, Z \subseteq X \). Thus, for every \( y \in Y \), \( \inf_{z \in Z} d(y, z) \) yields the distance to the element in \( Z \) that is closest to \( y \) (if there is such an element, otherwise the infimum is taken). Then, \( \sup_{y \in Y} \inf_{z \in Z} d(y, z) \) describes the largest minimal distance of elements \( y \in Y \) to elements \( z \in Z \). Note that the Hausdorff distance \( h^d \) is in general not symmetric, even if \( d \) is. To cover empty sets, we define for \( f \) a function, \( \sup_{x \in \emptyset} f(x) = 0 \) and \( \inf_{x \in \emptyset} f(x) = \infty \).

Remark 4.1 Rather than being metrics, the distances we use here are quasi-pseudo metrics: we do not require symmetry (i.e. \( d(x, y) \neq d(y, x) \)), and distinct elements may have distance 0 (i.e. \( d(x, y) = 0 \Rightarrow x = y \)). We use the word distance for simplicity.

Our metrics are not symmetric for the following reason. For a distance function \( d \) between a system implementation \( I \) and its specification \( S \), a distance \( d(I, S) \leq x \) expresses that for all behaviors \( \sigma \) of \( I \), there is a behavior \( \sigma \) of \( S \) at distance at most \( x \); i.e. deviations of at most \( x \) are allowed. It is not reasonable to expect that \( d(S, I) \leq x \) as well, since \( S \) may allow implementation freedom that has been resolved in \( I \). (Note that here, the distance between \( S \) and \( I \) is obtained as a Hausdorff distance on behaviors.)

Similarly it is reasonable that two different QTSs \( Q \) and \( Q' \) are at distance 0: if \( Q \) and \( Q' \) are isomorphic, then \( Q \neq Q' \), but we should have \( d(Q, Q') = 0 \) since the behaviors of \( Q \) and \( Q' \) are the same.

Given a distance function \( d \) on \( X \) and a set \( Y \subseteq X \), the \( \epsilon \)-ball \( B^d(Y, \epsilon) \) around \( Y \) contains all elements within distance \( \epsilon \) from some element in \( Y \). Formally, we define \( B^d(Y, \epsilon) = \{ x \in X \mid \exists y \in Y : d(x, y) \leq \epsilon \} \). For \( Y, Z \subseteq X \), set inclusion can be expressed as \( Y \subseteq Z = \forall y \in Y : \exists z \in Z : y = z \). A natural generalization of set inclusion is \( Y \subseteq^c Z = \forall y \in Y : \exists z \in Z : d(y, z) \leq \epsilon \). It is straightforward to show that \( Y \subseteq^c Z \) if and only if \( h^d(Y, Z) \leq \epsilon \). The following lemma gives a characterization of \( \subseteq^c \) in terms of (ordinary) set inclusion.

Lemma 4.2 Let \( d : X \times X \to \mathbb{R} \) be a distance and \( Y, Z \subseteq X \). Then \( Y \subseteq^c Z \) if and only if \( Y \subseteq B^d(Z, \epsilon) \).

4.2 Action and trace distances

The distances we will use in the following are action distances, trace distances, and their generalization to Hausdorff distances.

Figure 1 (b) allows for different approaches to testing a quantitative system.

One view is to see the implementation together with the perturbations inside a black box, which makes it impossible to know how large \( \gamma \) and \( \delta \) are. However, the testing objective here is to find out if the complete black box conforms, i.e. if the deviations seen in output are within the tolerated limits. In this scenario the tester would send inputs that are correct according to the specification, observe outputs that are sent back, measure the deviation of the received to the expected outputs according to the specification, and base its verdict on these deviations.

Another scenario is to assume that the tester has actually unperturbed access to the implementation itself. However, the implementation might be deployed in an environment in which inputs and outputs are perturbed by \( \delta \). The testing objective might then be to find out how the implementation reacts to perturbations in the input. This would require that the tester sends inputs to the implementation that are deliberately perturbed and deviate from the inputs prescribed by the specification. By testing it could then, for example, be established that a perturbation of inputs by at most \( \delta \) causes the implementation to produce outputs that are deviating by more than \( \delta \) (which could be seen as a reason to fail the test).

We show that both scenarios can be described in a single theory, and it is the choice of the trace distance [5] which makes the difference. For that reason we keep the definition of \( \mathbf{qicoqo} \) parametric, i.e. define a \( \mathbf{qicoqo}^D \), where \( D \) is the trace distance used to measure deviations in quantitative information. In the following we introduce two distances, corresponding to the two scenarios sketched above.

For our purposes, distances take values \( x \in [0, 1] \cup \{ \infty \} \). The \( \infty \) element is used to express incomparability between actions. To define the trace distances, we define first distances on (sets of) actions and lift these on the set of traces. In general, the distance between sets that we use here are Hausdorff distances.

Definition 4.3 (Action Distances) We define action distances \( ad^I, ad^O, ad^c, \) and \( ad^O \). Let \( \dagger \in \{ I, O \} \). Then

1. \( ad^I \) is defined as
   \[
   ad^I(a,b)(\gamma) = \begin{cases} 
   |x - y| & \text{if } a = b \text{ and } \{a,b\} \subseteq L^I, \\
   0 & \text{if } a = b \text{ and } \{a,b\} \not\subseteq L^I, \\
   \infty & \text{otherwise}.
   \end{cases}
   \]

2. \( ad^I \) (the constrained action distance), is defined as
   \[
   ad^I(a,b)(\gamma) = \begin{cases} 
   |x - y| & \text{if } a = b \text{ and } \{a,b\} \subseteq L^I, \\
   0 & \text{if } a(x) = b(y), \\
   \infty & \text{otherwise}.
   \end{cases}
   \]

All distances derived from \( ad^I \) are marked with subscript \( \cdot^c \).

3. For \( d \in \{ ad^I, ad^c \} \), \( E, E' \subseteq A \):
   \[
   d(E, E') = \sup_{a \in E} \inf_{b \in E'} d(a,b).
   \]

The action distance \( ad^O \) and \( ad^c \) measure the distances between output actions: for \( o(x), o(y) \in A_o \):

\[
ad^O(o(x), o(y)) = \sup_{a \in E} \inf_{b \in E'} d(a,b).
\]

They differ in the way how \( input \) actions are compared: for \( (x), i(y) \in A_i \), we set \( ad^O(i(x), i(y)) = 0 \), regardless of the values of \( x, y \). The distance \( ad^c \) is more constrained (thus the name): \( ad^c(i(x), i(y)) = 0 \) only if \( x = y \), and \( \infty \) otherwise. The same holds dually for \( ad^I \) and \( ad^c \). Note that all action distances result in \( \infty \) if the labels of the compared actions differ. For \( Y = \{o(x), i(y)\}, Z = \{o(x'), i(y')\} \)
with $y \neq y'$ it holds that $ad^O(Y, Z) = |x - y|$, whereas $ad^O(Y, Z) = \infty$.

We extend action distances to trace distances as follows.

**Definition 4.4 (Trace Distances)**

1. For traces $\sigma = \alpha_1 \cdots \alpha_n$, $\rho = \beta_1 \cdots \beta_m$, and $d \in \{I, O\}$, we define
   
   $td^d(\sigma, \rho) = \begin{cases} \max_{1 \leq i \leq n} ad^d(\alpha_i, \beta_i) & n = m \\ \infty & \text{otherwise.} \end{cases}$

   Moreover, $td(\sigma, \rho) = \max\{td^I(\sigma, \rho), td^O(\sigma, \rho)\}$.

2. For $d \in \{td^I, td^O, td\}$, and $P, Q$ QTS, we define the constrained trace distances, $td^d$ and $td^c$, are defined like $td^I$ and $td$, respectively, with $ad^c$ taking the place of $ad^I$.

3. The constrained trace distances, $td^I$ and $td^c$, are defined like $td^I$ and $td$, respectively, with $ad^c$ taking the place of $ad^I$.

The trace distances which we will consider in this paper are $td$ and $td_d$, where $td_d$ corresponds to the first scenario described above, and $td$ the second. We will let the variable $D$ range over $\{td, td_d\}$, if not indicated otherwise. The relation between these two distances is established in the following lemma.

**Lemma 4.5** Let $\sigma, \rho \in A^*$. Then $td_d(\sigma, \rho) \leq \varepsilon \land td^d(\sigma, \rho) \leq 0$ iff $td_d(\sigma, \rho) \leq \varepsilon$.

We define the set of states that can be reached from a starting state with a trace that lies within distance $\varepsilon$ from a given trace $\sigma$. The definition is generic for $D \in \{td, td_d\}$.

**Definition 4.6** Let $Q = (S, S^0, L, \rightarrow)$ be a QTS, and $D \in \{td, td_d\}$. Then, for $s \in S$, $\sigma \in A^*$ and $\varepsilon \in [0, 1]$. We define

$s_{after}^D(\sigma) = \{s' \mid \exists \rho \in A^*: s \xrightarrow{\rho} s' \land D(\sigma, \rho) \leq \varepsilon\}$.

For $S' \subseteq S$ we set $S'_{after}^D(\sigma) = \bigcup_{s \in S'} s_{after}^D(\sigma)$. We define $Q_{after}^D(\sigma) := S^0_{after}^D(\sigma)$.

5. **IMPLEMENTATION RELATIONS**

5.1 **Fuzzy trace inclusion**

A frequently used formal correctness criterion for an implementation w.r.t. to a specification is to demand that every trace of the implementation is also a trace of the specification. Implementation relations for non-quantitative transition systems with inputs and outputs (a la iocnf, ioco and the I/O refusal relation) can all be formulated in terms of trace inclusion. A natural adaption of this idea to quantitative systems is to replace strict set inclusion, $\subseteq$, with the quantitative version defined in Section 4.1. This idea leads to the following definition.

**Definition 5.1** We assume a QTS $S$ as specification and a QTS $I$ as implementation. We assume both $I, S$ being input-enabled. For $0 \leq \varepsilon \leq 1$ and $D \in \{td, td_d\}$, we define $I_{\varepsilon} \subseteq^D S$ iff $D(I, S) \leq \varepsilon$.

Thus, we define $I \subseteq^D S$ as $tr(I) \subseteq^D tr(S)$, and we obtain by Lemma 4.2 that $I \subseteq^D S$ iff $tr(I) \subseteq B^P(tr(S), \varepsilon)$. If $\varepsilon = 0$, then $\subseteq^D$ reduces to trace inclusion. Note that $\subseteq^D$ for $\varepsilon \neq 0$ is not a preorder, since transitivity does not hold: from $P \subseteq^D Q$ and $Q \subseteq^D R$ we cannot conclude that $P \subseteq^D R$. However, the triangle inequality that holds for $D$ allows us to conclude that $P \subseteq^D R$.

With the following lemma we get a different characterization of $\subseteq^D$.

**Lemma 5.2** Let $S, I$ be two input-enabled QTS and $D \in \{td, td_d\}$. Then $I \subseteq^D S$ iff for all $\sigma \in A^*$:

$out(I after^D \sigma) \subseteq^D out(S after^D \sigma)$

5.2 **qioco$^D$**

The formulation of $\subseteq^D$ in terms of out-sets of implementation and specification allows us now to define a relation on QTS which corresponds to the ioco relation in the non-quantitative case. We assume again QTS $S$ and $I$, with $I$ input-enabled. The classical way to define the qualitative ioco relation is to require inclusion of out-sets not for all possible words $\sigma \in A^*$, but only for traces of the specification. In the quantitative case, this restriction is too sharp. Since the idea is to cut the implementation some slack ($\varepsilon$, to be exact), it is necessary to consider also traces that are at most $\varepsilon$ off from the set of traces of the specification. The idea is that a tester sends inputs that are prescribed by the specification to the IUT, and receives outputs that may or may not be off from the expected output in the specification. We will therefore restrict the set of considered traces to $B^P(tr(S), \varepsilon)$, i.e. to the traces that are at most $\varepsilon$ off from the trace-set of the specification.

**Definition 5.3** Let $I$ and $S$ be two input-enabled QTS and $D \in \{td, td_d\}$. Define $I_{\varepsilon} \subseteq^D S$ if $\forall \sigma \in B^P(tr(S), \varepsilon)$:

$out(I after^D \sigma) \subseteq^D out(S after^D \sigma)$

**Example 5.4** In Figure 2, we see $I$, the implementation, and $S$, the specification. From the starting state, we have outgoing transitions, all labeled with $i^?$. After input $i^?((0, 0)$, specification $S$ indicates that only output $o^0((0, 0)$ is correct: $tr(S) = \{i^?((0, 0)$-o$^0((0, 0)$). The implementation $I$ yields after inputs $i^?((x)$ with $x \in (y - 0.2, y$) output $o^y((y)$, for $y = 0.2, 0.4, 0.6, 0.8, 1.0$.

\footnote{For the sake of simplicity we do not bother to make $I$ input-complete and non-blocking on outputs.}
We have $I \text{ qico}^D S$ for all $\varepsilon \in [0, 1]$, because the only trace of length $1$ in $B^D(\text{tr}(S), \varepsilon)$ is $?i(0)$. We then have $\text{out}(I \text{ after}^D_\varepsilon ?i(0)) = \emptyset = \text{out}(S \text{ after}^D_\varepsilon ?i(0))$, i.e. the delivered output coincides exactly with the expected one. However, $I \text{ qico}^D S$ only for $\varepsilon \in \{0, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The reason for this is that $B^D(\text{tr}(S), \varepsilon)$ contains $\{?i(x) \mid x \in [0, 1]\}$, and for, e.g., $\varepsilon = 0.1$ and $?i(0.05) \in B^D(\text{tr}(S), 0.1)$, $\text{out}(I \text{ after}^D_\varepsilon ?i(0)) = \emptyset = (0(0.2))$. $\text{td}(0(0.2), 0(0.0)) = 0.2 > \varepsilon$, which implies that conformance is not given. Only for the six given values that deviation allowed in inputs matches the maximal deviation in outputs.

5.3 qico$^D$ expressed as trace inclusion

It is a folklore result that qico-conformance coincides with trace inclusion if, apart from the implementation $I$, also the specification $Q$ is input enabled. The same is true in the quantitative case.

Theorem 5.5 Let $I$ and $S$ be input-enabled QTSs with the same action signature. Then $I \text{ qico}^D S$ iff $D(I, S) \leq \varepsilon$.

This result allows us to express qico$^D$ in terms of trace inclusion, based on demonic completion. Following [16], the idea is to manipulate the specifications such that they become input-enabled, yet retaining basically all the information w.r.t. their under-specification. For this to work we must assume that the considered QTS have a certain structure (are “well-formed”).

Definition 5.6 (well-formedness) Let $Q = \langle S, S^0, L, \rightarrow \rangle$ be a QTS (not necessarily input-complete). We say that $Q$ is well-formed, iff $\forall a \in A^* : s, s' \in Q$ after$^D_\sigma$ implies $\forall a \in A^* : s \rightarrow^a s' \rightarrow^a$.

Note that a well-formed QTS is not necessarily deterministic. Obviously, all deterministic QTSs are well-formed.

Definition 5.7 ($\Gamma$-Closure) Let $Q = \langle S, S^0, L, \rightarrow \rangle$ be a well-formed QTS. We define the $\Gamma$-closure of $Q$ as the QTS $\Gamma(Q) = \langle S', S^0, L, \rightarrow'^* \rangle$, where $S' = S \cup \{ \sigma r \}, \sigma r \not\in S$, and $\rightarrow'^* = \{ (s, a, s') \mid a \in A, s \rightarrow'^* s' \} \cup \{ (s', a, s') \mid a \in A \}$.

We call $\Gamma(Q)$ the $\Gamma$-closure of $Q$, and call $\sigma r$ the garbage collector (thus the $\Gamma$). Note that $\Gamma(Q)$ is input-enabled.

The definition of qico$^D$ uses the set $B^D(\text{tr}(S), \varepsilon)$. To express qico$^D$ in terms of trace inclusion, we must assume the existence of a QTS $B^D(S)$ such that $\text{tr}(B^D(S)) = B^D(\text{tr}(S))$.

Definition 5.8 Let $Q = \langle S, S^0, L, \rightarrow \rangle$ be a QTS. Then we denote by $B^D_Q$ the QTS $(S, S^0, L, \rightarrow^a)$, where $a \rightarrow^a s \subseteq S' \times A \rightarrow^a s'$ is the smallest set fulfilling the following property: $s \rightarrow^a s'$ implies $s^a \rightarrow^a s'$ for all $\beta \in A$ with $D(\alpha, \beta) \leq \varepsilon$.

Lemma 5.9 $\text{tr}(B^D_Q(S)) = B^D(\text{tr}(S), \varepsilon)$.

Now we can characterize qico$^D$ in terms of trace inclusion.

Theorem 5.10 Let $I$ be an input-enabled QTS and $S$ a well-formed one. Then

$I \text{ qico}^D S$ iff $\text{tr}(I) \subseteq \Gamma(I(S))$.

5.4 The qico$^D$ distance

The definition of the qico$^D$ relation in Section 5.2 is dissatisfactory in the sense that, for given $I$ the implementation and $S$ the specification, it lacks an indication of the minimal $\varepsilon$ such that $I \text{ qico}^D S$. It would be desirable to have a distance function $d^D_{\text{qico}}$ which actually measures the distance between $I$ and $S$. This function can be defined readily enough.

Definition 5.11 ($d^D_{\text{qico}}$) Let $I$ be an input-enabled QTS and $S$ a QTS. Then we define:

$$d^D_{\text{qico}}(I, S) = \inf \{ \varepsilon \in [0, 1] \mid I \text{ qico}^D S \}.$$ 

The following result extends Theorem 5.5 to the qico$^D$-distance.

Theorem 5.12 Let $I$ and $S$ be input-enabled QTSs with the same action signature. Then $d^D_{\text{qico}}(I, S) = D(I, S)$.

A different formulation of the above definition sheds light on how we can approximate $d^D_{\text{qico}}$ by means of testing. Another way to formulate $d^D_{\text{qico}}$ is as follows:

$$d^D_{\text{qico}}(I, S) = \sup \{ \varepsilon \in [0, 1] \mid \forall \varepsilon' < \varepsilon : I \text{ qico}^D S \}.$$ 

Using Lemma 5.10, this can be transformed to

$$\sup \{ \varepsilon \in [0, 1] \mid \forall \varepsilon' < \varepsilon : \text{tr}(I) \cap \Gamma(\text{tr}(S)) \neq \emptyset \}.$$ 

Thus for all $\varepsilon < d^D_{\text{qico}}(I, S)$, $\text{tr}(I) \cap \Gamma(\text{tr}(S)) \neq \emptyset$, i.e. $\exists \varepsilon' \in \text{tr}(I)$ which is not element of $\Gamma(\text{tr}(S))$. A testing approach to approximate $d^D_{\text{qico}}(I, S)$ is then the following: we start with $\varepsilon = 0$ and begin to synthesize a trace of the implementation by exchanging inputs and outputs between tester and implementation. Whenever we encounter a trace $\sigma \in \text{tr}(I)$ with $\sigma \not\in \Gamma(\text{tr}(S))$ we can conclude that the chosen $\varepsilon$ was too small. We must then derive an $\varepsilon'$ from $\varepsilon$ such that $\sigma \in \Gamma(\text{tr}(S))$. With this new $\varepsilon'$ we start testing from the beginning and synthesize another trace $\sigma'$, which gives us an $\varepsilon''$, and so on. In this way we approximate $d^D_{\text{qico}}(I, S)$. In the next section we will show how this general idea can be formulated in an on-the-fly testing algorithm.

6. ON-LINE TESTING

In this section we present a on-the-fly testing algorithm to approximate the qico$^D$ distance between an input-enabled QTS $I$ and a QTS $S$ by means of testing.

6.1 Stepwise distance measuring

To make the behavior of the implementation more accessible, we introduce the concept of trace functions.

Definition 6.1 (Trace function) Let $I$ be a QTS, input-enabled. A trace function $i$ of $I$ is a function $i : \text{tr}(I) \rightarrow A_0$ with the property $i(\sigma) = a$ implies $\sigma \cdot a \in \text{tr}(I)$. The set of all trace functions of $I$ is denoted as $TF(I)$.

If $\sigma \in \text{tr}(I)$, and $i \in TF(I)$, then $i(\sigma) = \text{out}(I \text{ after}_\sigma)$. A trace function thus picks one output from several and thus resolves the nondeterminism in outputs of $I$ after the execution of $\sigma$. Different executions of $I$ are described by different trace functions. Since $I$ is non-blocking on outputs, $i$ is total.
In the following, we will use the trace functions \( i \in TF(\mathcal{I}) \) to represent the behavior of \( \mathcal{I} \). The following definition describes a way to express the distance of trace \( \sigma = \alpha_1 \alpha_2 \cdots \alpha_n = \mathcal{D}(\sigma, tr(\mathcal{S})) \), stepwise in terms of \( \alpha_1, \alpha_2, \ldots, \alpha_n \).

**Definition 6.2** Let \( S = (S, S^0, L, \rightarrow) \) be a QTS, \( i \in TF(\mathcal{I}) \) and \( D \in \{td, td^i\} \). We define for \( S \) and \( i \) a family of functions, \( curr_{\text{dist}}^D : S \rightarrow [0, 1]_\infty \) with \( \sigma \in A^* \), \( i(\sigma) \) as follows. (1) \( curr_{\text{dist}}^D(\sigma) = 0 \) if \( \sigma \in S^0 \), and \( \infty \) otherwise; (2) for \( \alpha = i(\sigma) \) or \( \alpha \in A_i \): \( curr_{\text{dist}}^D(s) = \inf_{s' \rightarrow s} \max\{curr_{\text{dist}}^D(s'), D(\alpha, b)\} \).

Then \( curr_{\text{dist}}^D(s) \) is the minimal trace distance w.r.t. \( D \) of a trace \( \sigma \) from the set of traces \( \{\rho \in A^* | \exists \sigma_0 \in S^0 : \sigma_0 \sigma = s\} \), as is stated in Theorem 6.3.

**Theorem 6.3** \( curr_{\text{dist}}^D(s) = \mathcal{D}(\sigma, \rho) \).

For a more convenient construction of the \( curr_{\text{dist}} \) functions in the algorithm to come, we introduce the operator \( C : (S \rightarrow [0, 1]_\infty) \times A \times \{td, td^i\} \rightarrow (S \rightarrow [0, 1]_\infty) \) as follows:

\[
C(c, \alpha, D) = s \mapsto \inf_{s' \rightarrow s} \max(c(s'), D(\alpha, b))).
\]

Clearly, \( C(curr_{\text{dist}}^D, \alpha, D) = curr_{\text{dist}}^D \).

6.2 The algorithm

The algorithm for on-the-fly testing of QTS has two parts. The first is the actual testing algorithm which synthesizes a trace of the implementation and measures the distance of this trace to the specification. The second algorithm uses \( curr_{\text{dist}} \) to construct the behavior of the implementation in this particular test run. \( n \) is the maximal number of test steps to be executed, and is chosen arbitrarily. \( D \in \{td, td^i\} \) is the distance function to be used. Finally, \( \varepsilon \in [0, 1] \) is a tolerance parameter which has influence on the inputs to be chosen to trigger the implementation. \( MQOTF \) returns a tuple \( (cd, \sigma) \), where \( \sigma \in tr(\mathcal{I}) \) is the trace which was generated during testing, and \( cd \in [0, 1]_\infty \). Later we will show that \( cd = \max(\varepsilon, D(\sigma, tr(\mathcal{S}))) \). The main purpose of \( MQOTF \) is to construct the function \( curr_{\text{dist}} \) step-by-step, where \( \sigma \) is the trace synthesized during testing.

In lines 2–5, several local variables are initialized: \( \sigma \) is the trace observed so far, and is initialized with \( \lambda \). \( cd \) keeps track of the lower bound of the distance of the observed trace to \( tr(\mathcal{S}) \) and is initialized with parameter \( \varepsilon \). \( curr_{\text{dist}} \) is the current \( curr_{\text{dist}} \) function and is initialized with \( curr_{\text{dist}}^\lambda \). \( M \) is the so-called menu, the set of states of \( \mathcal{S} \) which can be reached with traces \( \rho \in tr(\mathcal{S}) \) such that \( D(\sigma, \rho) \leq cd \). \( M \) is initialized with the initial states of \( \mathcal{S} \).

Lines 6 to 18 cover the main loop of \( MQOTF \), which is terminated if \( cd = \infty \) or \( |\sigma| > n \). The execution starts either on line 7 or 12. On line 7, an input \( \alpha \in A_i \) is chosen such that \( M_{\text{after}}(\alpha) \neq \emptyset \). If such an \( \alpha \) exists, \( curr_{\text{dist}} \) is updated, new menu \( M \) is defined, and \( \alpha \) is appended to \( \sigma \) (lines 8–10). Note that \( cd \) is not updated, since \( \sigma \cdot \alpha \) has the same trace distance to \( tr(\mathcal{S}) \) as \( \sigma \). This is ensured by the condition on the choice of \( \alpha \) on line 7. If execution continues with line 12, rather than 7, the output \( i(\sigma) \) is used to update \( curr_{\text{dist}} \), \( cd \), and \( M \) and \( \sigma \). Note that \( cd \) is only increased if \( D(\sigma \cdot \alpha, tr(\mathcal{S})) \) is larger than \( \varepsilon \). Once the while-loop terminates, line 19 is reached. The computed distance \( cd \), together with \( \sigma \) is then returned.

\( MQOTF \) returns \( cd, \sigma \), i.e. the trace distance of one trace only. Assuming that \( \varepsilon = D(\sigma, tr(\mathcal{S})) \) (this is shown in Section 6.3), \( MQOTF \) can be used to approximate \( d_{\text{approx}}(\mathcal{I}, S) \), as it has been sketched in Section 5.4 and is worked out in Algorithm 3. There, we have again a number \( n \in \mathbb{N} \), which bounds the number of test runs to be executed and which is chosen arbitrarily. Moreover, we have the usual \( \mathcal{I}, S \) and \( D \). The approximation takes place in the while-loop between lines 5 and 7. In each run through the loop, an \( m \in \mathbb{N} \) is chosen, which is used to restrict the length of the test run. Moreover, a trace function \( i \in TF(\mathcal{I}) \) is chosen nondeterministically from \( TF(\mathcal{I}) \). This choice reflects the fact that in each test run the implementation \( \mathcal{I} \) might actually behave differently from a previous test run, even if the same inputs are applied. \( MQOTF \) is called with the current value of \( cd \) as tolerance parameter, initially 0. The value of \( cd \) is constantly updated with the distance computed by \( MQOTF \).

6.3 Soundness and completeness of MQOTF

Algorithm 2 is sound w.r.t. \( qic_0 \), for \( D \in \{td, td^i\} \). Soundness means that, whenever \( \mathcal{I} qic_0 \) is sound for all \( n \in \mathbb{N}, i \in TF(\mathcal{I}) \) and possible return values \( cd, \sigma \) from \( MQOTF(i, S, n, D, \varepsilon) \), \( cd = \varepsilon \) holds. The algorithm is also complete, i.e. if \( \mathcal{I} qic_0 \) is sound, then there is a trace function \( i \in TF(\mathcal{I}) \) and a run procedure \( MQOTF(i, S, n, D, \varepsilon) \) with return value \( cd, \sigma \) such that \( cd > \varepsilon \).

Integral part of a soundness proof is to show that the fol-
Algorithm 3 Approximating dist_

Require: S = (S, S^0, L, \rightarrow) is a QTS, T an input-enabled QTS, n \in \mathbb{N}, D \in \{t, td^D\}.

1: n' \leftarrow 0
2: cd \leftarrow 0
3: \sigma \leftarrow \lambda
4: while n' \leq n and cd < \infty do
5: \text{[true]} \rightarrow let i \in TF(T), m \in \mathbb{N} in
6: \text{(cd, } \sigma) \rightarrow \text{mqotp}(i, S, m, D, cd)
7: n' \leftarrow n' + 1
8: end
9: end while

lowing property of Algorithm 2 holds: whenever execution reaches line 6 it holds: (1) curr_dist = curr_dist^D; (2) cd = max{D(\sigma, tr(S)), \epsilon}; (3) M = \{s | curr_dist(s) \leq cd\}; (4) |\sigma| \leq n + 1.

These conditions are easily verified when line 6 is entered for the first time. Then \sigma = \lambda, cd = \epsilon (D(\lambda, tr(S)) = 0), curr_dist = curr_dist^D, M = S^0 = \{s | curr_dist(s) = 0\}, and |\sigma| = 0. If we assume that all four condition hold and additionally M \neq 0 and |\sigma| \neq n + 1, the loop body is entered, and a non-deterministic choice has to be made on either to continue with line 7 or line 12. If the pre-condition of line 7 holds and the line is nondeterministically chosen, then action a' \in A_T is the input selected to be sent to the implementations (which is only implicitly done by appending a? to \sigma). In line 8, curr_dist is updated. From the definition of C it is easy to see that then curr_dist = curr_dist^D at line 9. Important to note is that in lines 8-10 the value of cd is not updated. The reason is that in fact inf_{s \in S} curr_dist^D(a'(s)) = inf_{s \in S} curr_dist^D(s), since the input a'? is chosen to not deviate more than \epsilon from the specified inputs. The trace distance of \sigma \cdot a'? to S is therefore equal to that of \sigma. When we return from line 10 to line 6, the four conditions are thus still satisfied.

If line 12 is chosen, output a! is received from the implementation (symbolized by consulting the trace function). In line 13, curr_dist is updated from curr_dist^D to curr_dist^D at line 13, cd is updated. By the precondition and Theo-rem 6.3, then cd = max{\max(\epsilon, D(\sigma, tr(S))), D(\sigma \cdot a!, tr(S))} = \max(\epsilon, D(\sigma \cdot a!, tr(S))). In the remaining lines until line 16, the remaining variables are updated. Clearly, on return to line 6, the four conditions hold again.

The fact that these conditions hold also once line 19 is reached allows the conclusion that, once mqotp returns a result (cd, \sigma), then cd = max{\epsilon, D(\sigma, tr(S))}.

To prove now soundness, we assume that I for some \sigma \in S, \sigma^a \in S, then that a run of mqotp(i, S, n, D, e) for i \in TF(T) returns (cd, \sigma) with cd > \epsilon. We know then that cd = D(\sigma, tr(S)). Then there is also a prefix \sigma^a of \sigma such that D(\sigma^a, tr(S)) \leq \epsilon, but D(\sigma^a \cdot a!, tr(S)) > \epsilon (only outputs can increase the distance of a trace to tr(S)). Then \sigma^a \in B^D(tr(S), e), and a! \in out(I after^D \sigma). However, this implies also that out(I after^D \sigma) \not\subseteq out(S after^D \sigma), a contradiction to the assumption I. 

To show completeness we have to prove that, if I for some \sigma \in S, then there is a trace function i \in TF(T) and a run of procedure mqotp(i, S, n, D, e) with return value (cd, \sigma) such that cd > \epsilon. I implies according to the definition of qioco^D that there is a \sigma \in B^D(tr(S), e) with out(I after^D \sigma) \not\subseteq out(S after^D \sigma). There is thus an output a! \in out(I after^D \sigma) with D(\sigma^a \cdot a!, tr(S)) > \epsilon, and moreover, D(\sigma^a \cdot a!, tr(S)) > \epsilon. This implies that \{\sigma \cdot a! \subseteq tr(S), i.e. there is also a trace function i \in TF(T) with i(\sigma) = a!. Let n = |\sigma|. Since \sigma \in B^D(tr(S), e), we can assume that there is a run through mqotp(i, S, n, D, e) such that we enter line 7 of Algorithm 2 with the following conditions fulfilled: (1) curr_dist = curr_dist^D; (2) cd = \epsilon \geq D(\sigma, tr(S)); (3) M = \{s | curr_dist(s) \leq \epsilon\}; (4) n' = n. If the algorithm proceeds then to line 12, trace function i will return output a!, curr_dist will be updated to curr_dist^D and cd to max{\epsilon, inf_{s \in S} curr_dist(s)} = D(\sigma \cdot a!, tr(S)). Thus cd > \epsilon. Since n' will be updated to n + 1, the algorithm will terminate and return with (cd, \sigma \cdot a!), where cd > \epsilon. This was to be shown.

7. OFF-LINE TESTING

This section presents an off-line approach to quantitative testing. That is, we explain how one can derive test cases from a QTS, how these test are executed on an IUT and how the results are evaluated. We show that the off-line framework is sound and complete and present the connection with the on-the-fly approach from the previous section.

It turns out that defining test cases for input-enabled specifications is possible in a remarkably effortless way. However, we only consider input-enabled specifications: leaving the extension to specifications that are not input-enabled for future research. Also, we only consider the trace distance td, i.e. we take D = td. Since dist^D and the trace distance td coincide for input-enabled systems, we will work td as the implementation relation.

7.1 Test cases

We consider test cases that are adaptive, i.e. the next action to be performed (observe the IUT, stimulate the IUT or stop the test) may depend on the test history, that is, the trace observed so far. If, after a trace \sigma, the tester decides to stimulate the IUT with an action a'?, then the new test history becomes \sigma a'?; in case of an observation, the test accounts for all possible continuations \sigma a'! with a'! \in L^O an output action. Ioco theory requires that tests are "fail fast", i.e. stop after the discovery of the first failure, and never fail immediately after an input. Formally, a test case t consists of the set of all possible test histories obtained in this way. Alternatively, we can represent each test case as a QTS S_\sigma, which in each state either selects one input action, or enables all output actions.

Definition 7.1 A test case (or test) t for S is a prefix-closed subset of \epsilon such that, (1) if a? \epsilon \in t, then \sigma a'? = t for any \beta \in A with a'? \neq \beta, (2) if a! \epsilon \in t, then \sigma a(! \in t for all \beta \in A_0, (3) if \sigma \notin tr(S), then last(\sigma) \in A_0 and \sigma is no proper prefix of any \sigma' \in t, and (4) t does not contain any strictly increasing chain \sigma_0 < \sigma_1 < \sigma_2 < \ldots .

The leaves of t, denoted leaves(t), are those \sigma \in t which are not a proper prefix to any \sigma' \in t. We denote the set of all tests for S by TESTS(S).

The following lemma states that every behavior of the specification S can be tested.

Lemma 7.2 For all \sigma \in tr(S), there is a test t \in TESTS(S) such that \sigma \in t.
Any test case can be represented by a deterministic, tree-shaped QTS, whose traces are exactly the traces of $t$. By abuse of notation, we often write $t$ for $S_t$.

**Definition 7.3** Let $t$ be a test for QTS $S$. The QTS-representation of $t$ is the QTS $S_t = (S_t, S_0^t, L_t, \rightarrow_t)$ given as follows. The states are all traces in $t$, i.e. $S_t = t$; the initial state is the empty trace, i.e. $S_0^t = \{\lambda\}$; its labels are exactly the labels of $S$, i.e. $L_t = L$; and the transition relation $\rightarrow_t \subseteq S_t \times A_t \times S_t$ is given by $\{(s, a, sa) \mid sa \in t\}$.

It immediately follows that $tr(S_t) = t$.

**Example 7.4** Figure 3 shows the specification $S_{\text{coff}}$ of a coffee machine, where the user inputs the strength of the coffee (in $[0,1]$) and then should get a coffee of the desired strength. Note that the picture only shows a skeleton of an infinite QTS, the idea being that an implementation fails a test case $t$ on implementation $I$ if and only if $tr(S_t) = t$.

We interpret a test case quantitatively, i.e. rather than a pass or a fail, our verdict function assigns a number in $[0,1]$ to each leaf of a test case.

**Definition 7.5** Let $t$ be a test for QTS $S$. The quantitative verdict function $v_S$ is the function $v_S : leaves(t) \rightarrow [0,1]$, with $v_S(\sigma) = td(\sigma, tr(S))$. We call the pair $q_t = (t, v_S)$ a evaluated test for $S$, and $ET(S)$ the set of all evaluated tests.

The following result shows that one can test any behavior with a finite distance to a specification.

**Lemma 7.6** For all $\sigma$ with $td(\sigma, S) = \delta \leq 1$, there exists an evaluated test $(t, v) \in ET(S)$ such that $\sigma \in t$ and $v(\sigma) = \delta$.

### 7.2 Test execution

As in the qualitative case, tests are executed by composing them in parallel with the IUT. To accommodate imprecision, we employ an imprecise parallel composition operator. The idea is as follows. Tests describe the intended, precise behavior. However, due to imprecisions, deviations from the desired behavior may occur when we execute the test case on an IUT: we may want to stimulate the IUT with action $a'\in QTS S$ and action $b\in QTS I$ may synchronize with any action $a$ in $I$ within action distance $\delta$.

This is formalized by the imprecise parallel composition operator $\parallel$.

**Definition 7.7** For two QTS $Q$ and $P$ be two QTS with the same action signatures. Let $\delta \geq 0$. We define the parallel composition with tolerance $\delta$, denoted $Q\parallel^\delta_P$, as the QTS $(S, S_0^t, L_\parallel^t, \rightarrow\parallel^t)$ given by $S_{Q\parallel^\delta_P} = S_Q \times S_P$ and $S_0_{Q\parallel^\delta_P} = S_0^Q \times S_0^P$, and

- $L_{Q\parallel^\delta_P} = (L_Q, L_P)$,
- $\rightarrow_{Q\parallel^\delta_P} = \{(s, u)^{\alpha}Q\parallel^\delta_P(s', u') \mid s^{\alpha}Qs' \land u^{\alpha}P u'\}$.

Here, $\rightarrow_{Q\parallel^\delta_P}$ denotes the $\delta$-transition relation given by $s^{\alpha}Qs'$ iff there exists an $\alpha \in A_P$ with $ad(\alpha, \beta) \leq \delta$ and $s^{\alpha}P s'$.

Note that $\parallel$ is not symmetric since only the right component is allowed to deviate.

Suppose we run, with an imprecision of at most $\delta$, a test case $t$ on implementation $I$. Then the set of all possible executions are exactly the traces of $S_t \parallel I$.

**Definition 7.8** Let $t = (t, v) \in ET(S)$ be an evaluated test for $S$ and $T \subseteq ET(S)$ be an evaluated test suite for $S$. Let $\delta > 0$. The set of test executions is given by $exec^v(t, I) = tr(S_t \parallel I)$.

### 7.3 Test evaluation

In the qualitative case, an implementation fails a test case if at least one of the executions leads to a fail verdict; the implementation fails a test suite if at least one of the test cases fails. We also employ this worse case scenario: the quantitative verdict is the largest deviation that we may encounter during test execution.

**Definition 7.9** Let $t = (t, v) \in ET(S)$ be an evaluated test for $S$ and $T \subseteq ET(S)$ be an evaluated test suite for $S$. Let $\delta \geq 0$. The verdict of $v_{T}(I)$ is given by $v_{T}(I) = sup_{t \in T} v_{T}(t)$, and the verdict of $v_{\gamma}(I)$ is given by $v_{\gamma}(I) = sup_{t \in T} v_{\gamma}(t)$.

**Example 7.10** Figure 3 depicts an implementation $I_{\text{coff}}$ of a coffee machine, where the user always gets a coffee of strength $0.5$. Note that $td(I_{\text{coff}}, S_{\text{coff}}) = 0.5$. However, if we run the $I_{\text{coff}}$ against test $t$, then we obtain for $\delta = 0.1$ that $exec(t, I_{\text{coff}}) = \{\text{stren}(y)\text{coff}(x) \mid y \in [0.5,0.9], x \in [0.5,0.6]\}$. Thus, $v_{\gamma}(I_{\text{coff}}) = 0.4$, which is witnessed by the trace $\text{stren}(y)\text{coff}(0.5)$.

The following lemma is instrumental in proving the soundness and completeness result below.

**Lemma 7.11** Let $t = (t, v) \in ET(S)$ be an evaluated test for $QTS S$ and let $\delta \geq 0$.

1. $exec(t, I) = leaves(t) \cap B_\delta(tr(I))$.
2. $v_{\gamma}(I) = td(t \parallel I, S)$. 
7.4 Correctness of off-line testing

Soundness and completeness express the key correctness of the test framework: in the qualitative case, it shows that, for a specification $S$ any conforming implementation passes all tests derived from $S$ (soundness) and that for any non-conforming implementation, there is at least one test that exhibits the error, i.e. yields a fail (completeness). In the quantitative case, we prove that the worst case verdict that we obtain when we run all tests from $\text{TESTS}(S)$ against an implementation $I$ is exactly the trace distance, corrected with the imprecision $\delta$. Let $\gamma = t\delta(I, S)$.

Theorem 7.12 (soundness & completeness)

$$v^\delta_{\text{TESTS}(S)}(I) = \gamma + \delta.$$  

We show in the following the connection of the on-the-fly algorithm $\text{MQOTF}$ to the test execution of test cases. We need for that the following definition.

Definition 7.13 Let $Q = (S, S^0, L, \rightarrow)$ be a QTS, and $\delta \in [0, 1]$. We define $Q_\delta$ as QTS $(S, S^0, L, \rightarrow_\delta)$, where $s \rightarrow_\delta s'$ iff either $s \sim_\delta s'$ and $\alpha \in A_I$, or $s \sim_\delta s'$ and $\alpha' \in A_O$ with $\text{ad}(\alpha', \alpha) \leq \delta$.

Theorem 7.14 Let $I, S$ be input-enabled QTSs with the same action signature. Then

$$\sup_{i, n} \{cd | (cd, \sigma) \in \bigcup_{i, n} \text{MQOTF}(i, S, n, td, 0)\} = \gamma + \delta.$$  

Here $i$ ranges over the trace functions of $I_\delta$ (cf. Definition 7.13), and $n$ over the natural numbers.

8. CONCLUSIONS AND FURTHER WORK

We introduced an ioco-based metric on QTSs, which measures how far a system implementation lies from its specification. We also presented on-line and off-line test case derivation algorithms, which were shown to be sound and complete with respect to the metric. Working in a completely quantitative setting, also the test verdict is quantitative: rather than giving a pass/fail answer, the verdict estimates the distance (given by our metric) from the UUT to its specification.

Our framework lies down the semantical foundations for quantitative testing. For the algorithms to be effectively implementable, one needs to find finite, symbolic methods for representing and manipulating in efficient way the various objects playing a role in the testing process. In particular, we need finite representations for test cases, the function $\text{curr}_\\delta\text{dist}$ and efficient methods to compute the function $\text{after}_\\delta$.

The numerical information in the developed theory in uninterpreted. Thus, our theory is independent from any concrete semantic domain. An important topic to be addressed is therefore to integrate it into existing testing theories with concrete quantitative elements, like timed testing [1, 3] or hybrid testing [17].

9. REFERENCES