Quantitative Testing*

Henrik Bohnenkamp
Software Modeling & Verification
Department of Computer Science
RWTH Aachen University
D-52056 Aachen, Germany
henrik@cs.rwth-aachen.de

Mariëlle Stoelinga
Department of Computer Science
University of Twente
7500 AE Enschede
The Netherlands
marielle@cs.utwente.nl

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 ε", allowing deviations of implementation from the specification of at most ε. 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

Specification-based Testing, Conformance Relations, Test Case Generation, Test Execution, Robustness.

*This research has been partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.505 (MOQS); by the EU under grants numbers IST-004527 (ARTIST2) and FP7-ICT-2007-1 (QUASIMODO); and by the DFG/NWO bilateral cooperation program under project number DN 62-600 (VOSS2).

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-*

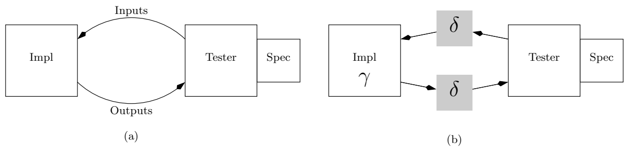

Figure 1: Testing Scenarios

tion systems (QTS). These are an extension of input/output transition systems with continuous information: Each action in a QTS carries also a value $x \in [0, 1]$. Based on this model class, we define conformance relations $\mathbf{qioco}_\varepsilon$, a conservative extension of the well-known *ioco* relation [14] and parameterized with a tolerance value ε. An implementation conforms to a specification as long as it is functionally correct (i.e. delivers only outputs that are expected) and deviates in the quantitative part by at most ε. The presented theory relies on so-called *distance functions* [5], or distances. These distances, defined on the actions, traces and QTS, measure how far one action, trace or QTS lies apart from another. Our testing scenario finds out how far an IUT is from conforming to the specification: We show that, if every output generated by the IUT lies closely to a output one expects, then the distance (formalized by a quantitative notion of the conformance relation $\mathbf{qioco}_\varepsilon$) will be small, otherwise it will be large.

We start out from the classical testing framework, as it is depicted in Figure 1 (a) and formalized in the *ioco* theory. The tester has access to the specification, and sends inputs derived from the specification to the implementation. The implementation responds with one or more outputs (or no output at all). The tester checks whether the received (lack of) output is correct according to specification.

For our quantitative testing framework, we assume that the specification and implementation can be modeled by QTS. Inputs are of the form $i?(x)$ and outputs of the form $o!(y)$[1]; i and o indicate the input or output event, and x, y the quantitative information assigned to it. In Figure 1 (b) we extend the previous scenario with two boxes which represent the perturbances inputs and outputs are subjected to. The sources of the perturbances are not relevant, but we assume that the perturbances are at most δ: an input $a?(x)$ sent to the implementation arrive at the IUT as input $a?(x')$, where $|x' - x| \leq \delta$. Similarly, the implementation may then produce an output $b!(y)$, that may arrive as $b!(y')$ at the tester, where again $|y - y'| \leq \delta$. The implementation itself might also be a source of deviation, where an input, even if unperturbed in transmission, might be interpreted as $a?(x')$ with $|x - x'| \leq \gamma$, and an output $b!(y)$ might be sen out as an output $b!(y')$ with $|y - y'| \leq \gamma$.

Based on this model and the relation $\mathbf{qioco}_\varepsilon$, we present an distance d_{qioco} that measures how far a system implementation lies from a system specification. We present two testing algorithms that estimate the d_{qioco}-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 subsequently executed against the IUT. Both approaches are sound and complete with respect to $\mathbf{qioco}_\varepsilon$, up to the perturbations of ε. This means that if $\varepsilon = \gamma + \delta$ and $\mathcal{I}\ \mathbf{qioco}_\varepsilon\ \mathcal{S}$, then no test case derived from $\mathcal{S}$ will report a distance of more than ε (soundness) and there exist a test case that gets arbitrary close to ε (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 *ioco* theory. In Section 3 we introduce QTS. In Section 4 we introduce distances on sets of traces. In Section 5 we define the $\mathbf{qioco}_\varepsilon$ relations and analyze some of their properties. In Section 6 we introduce the on-the-fly testing algorithm for $\mathbf{qioco}_\varepsilon$ 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 *ioco* testing theory [14], which are our starting point for the quantitative testing approach. Specification-based testing à la *ioco* is all about sequences of inputs and outputs, the so-called *traces*. The most interesting traces are the *test executions*, 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^I, L^O, are finite. A test execution is thus formally a trace $\sigma \in (L^I \cup L^O)^*$, which is synthesized 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 $T \subseteq (L^I \cup L^O)^*$. 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 σ must be element of T, $\sigma \in T$. The tester can choose between sending an input to the implementation and waiting for an output from the implementation. If tester and implementation have already composed test execution σ and the tester decides to continue testing by sending an input, it will only choose an input $i?$ such that $\sigma \cdot i? \in T$. A specification does not need to be input-enabled, i.e. it is allowed that $\{i? \mid \sigma \cdot i? \in T\} \subsetneq L^I$. The implementation *must* be input-enabled, i.e. must be able to accept all inputs at all times. The implementation extends a test execution σ by returning outputs. If it returns output $o! \in L^O$ and $\sigma \cdot o! \in T$, then testing can continue. If, however, $\sigma \cdot o! \notin T$, then this is considered by the tester as a test-failure, since the correctness criterion is violated. Testing stops in this case. This scheme does hinge on the requirement that an implementation always produces an output eventually, if the tester waits for one. This is however not realistic: consider a web server as implementation to be tested. Such a server would never produce an output after it is freshly started, before not some request (i.e. an input) for a web page comes in. The *ioco* testing approach considers therefore *quiescence* of the implementation. That means in practice that, if an implementation does not produce an output, the tester extends the test execution σ after a timeout of appropriately chosen length with a synthetic output δ, which somewhat

[1]We mark inputs with ?, outputs with !.

[2]We assume the tester to be a software tool.

Algorithm 1 *ioco* testing algorithm

```
Require: T set of traces of specification, I the implementation, n ∈ ℕ .
 1: procedure IOCO_OTF(I, T, n)
 2:     σ ← λ;
 3:     while |σ| ≤ n do
 4:         [ σ·i? ∈ T]→
 5:             send i? to I
 6:             σ ← σ·i?;
 7:         end
 8:         [true] →  receive output o! from I
 9:             σ ← σ·o!;
10:             if σ ∉ T return(fail)
11:         end
12:     end while
13:     return(pass)
14: end procedure
```

paradoxically denotes "*lack of output observed*". Therefore the previous statement that $T \subseteq (L^I \cup L^O)^*$ 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 $T \subseteq (L^I \cup L^O \cup \{\delta\})^*$. *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 $L^I \cup L^O$, 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 $o! \in L^O$: $\sigma{\cdot}o! \in I$ implies $\sigma{\cdot}o! \in T$. 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 $(L^I \cup L^O \cup \{\delta\})$. 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. This 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 σ denotes the current test execution and is initialized with λ (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, σ 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 write the concatenation of sequences $\sigma, \rho \in A^*$ by juxtaposition, i.e. as $\sigma\rho$.

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

Definition 3.1 *The tuple* $Q = \langle S, S^0, L, \rightarrow\rangle$ *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 a set of action labels, which is partitioned into two sets (L^I, L^O) of* input *and* output *labels respectively. We write $A_I = L^I \times [0, 1]$, $A_O = L^O \times [0, 1]$ 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 $s \xrightarrow{\alpha}$, if $\exists s' \in S : s \xrightarrow{\alpha} s'$. We denote by $out(s) = \{\alpha \in A_O | s \xrightarrow{\alpha}\}$ the set 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 $\langle S, S^0, L, \rightarrow\rangle$ to 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 δ-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^O with a label δ, and add to every state $s \in S$ which is blocking on outputs (i.e. without any outgoing output-transition) a transition $s \xrightarrow{\delta(0)} s$. 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 $s, s', s'' \in S, \alpha \in A$: $s \xrightarrow{\alpha} s'$ and $s \xrightarrow{\alpha} s''$ implies $s' = s''$; Q is* input-enabled *iff for all $s \in S$, $\alpha? \in A_I$ we have $s \xrightarrow{\alpha?}$.*

Definition 3.3 (Traces) *An* execution fragment *of Q is a finite sequence $\nu = s_0 \alpha_1 s_1 \alpha_2 s_2 \ldots s_n$ such that $s_{i-1} \xrightarrow{\alpha_i} s_i$ for all $1 \leq i \leq n$. The* trace *of ν is obtained by removing all states in ν, i.e. $trace(\nu) = \alpha_1 \alpha_2 \cdots \alpha_n$. We then write $s_0 \xrightarrow{\alpha_1 \alpha_2 \cdots \alpha_n} s_n$. We denote by $tr(Q) \subseteq A^*$ the set of all traces σ 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 $d : X \times X \to \mathbb{R}^{\geq 0}_{\infty}$, 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 to sets via the Hausdorff distance $h^d : \mathcal{P}(X) \times \mathcal{P}(X) \to \mathbb{R}^{\geq 0}_{\infty}$, 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 close to y (if there is such element, otherwise the infimum is taken). Then, $\sup_{y \in Y} \inf_{z \in Z} d(y,z)$ describes the 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. $d(x,y) = 0 \not\Longrightarrow x = y$). 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 $\mathcal{I}$ and its specification $\mathcal{S}$, a distance $d(\mathcal{I},\mathcal{S}) \leq x$ expresses that for all behaviors σ of $\mathcal{I}$, there is a behavior σ of $\mathcal{S}$ at distance at most x; i.e. deviations of at most x are allowed. It is not reasonable to expect that $d(\mathcal{S},\mathcal{I}) \leq x$ as well, since $\mathcal{S}$ may allow implementation freedom that has been resolved in $\mathcal{I}$. (Note that here, the distance between $\mathcal{S}$ and $\mathcal{I}$ 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 ε-ball $B^d(Y,\varepsilon)$ around Y contains all elements within distance ε from some element in Y. Formally, we define $B^d(Y,\varepsilon) = \{x \in X \mid \exists y \in Y : d(x,y) \leq \varepsilon\}$. 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 $Y \subseteq^d_\varepsilon Z \hat{=} \forall y \in Y : \exists z \in Z : d(y,z) \leq \varepsilon$. It is straightforward to show that $Y \subseteq^d_\varepsilon Z$ if and only if $h^d(Y,Z) \leq \varepsilon$. The following lemma gives a characterization of $\subseteq^d_\varepsilon$ in terms of (ordinary) set inclusion.

Lemma 4.2 *Let $d : X^2 \to \mathbb{R}$ be a distance and $Y, Z \subseteq X$. Then $Y \subseteq^d_\varepsilon Z$ if and only if $Y \subseteq B^d(Z,\varepsilon)$.*

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 systems.

One view is to see the implementation together with the perturbances inside a black box, which makes it impossible to know how large γ and δ 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 δ. 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 δ causes the implementation to produce outputs that are deviating by more than δ (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 $\mathbf{qioco}_\varepsilon$ parametric, i.e. define a $\mathbf{qioco}^{\mathcal{D}}_\varepsilon$, where $\mathcal{D}$ 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 $x \in [0,1]_\infty := [0,1] \cup \infty$. The ∞ 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^I_c, and ad^O_c. Let $\dagger \in \{I, O\}$. Then*

1. *$ad^\dagger$ is defined as*

$$ad^\dagger(a(x), b(y)) = \begin{cases} |x-y| & \text{if } a = b \text{ and } \{a,b\} \subseteq L^\dagger, \\ 0 & \text{if } a = b \text{ and } \{a,b\} \not\subseteq L^\dagger \\ \infty & \text{otherwise.} \end{cases}$$

2. *$ad^\dagger_c$ (the* constrained *action distance), is defined as*

$$ad^\dagger_c(a(x), b(y)) = \begin{cases} |x-y| & \text{if } a = b \text{ and } \{a,b\} \subseteq L^\dagger, \\ 0 & \text{if } a(x) = b(y), \\ \infty & \text{otherwise.} \end{cases}$$

All distances derived from $ad^\dagger_c$ are marked with subscript $\cdot_c$.

3. *For $d \in \{ad^\dagger, ad^\dagger_c\}$, $E, E' \subseteq A$:*
$$d(E,E') = \sup_{a \in E} \inf_{b \in E'} d(a,b).$$

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

$$ad^O(o(x), o(y)) = ad^O_c(o(x), o(y)) = |x-y|.$$

They differ in the way how *input* actions are compared: for $i(x)$, $i(y) \in A_I$, we set $ad^O(i(x), i(y)) = 0$, regardless of the values of x, y. The distance ad^O_c is more constrained (thus the name): $ad^O_c(i(x), i(y)) = 0$ only if $x = y$, and ∞ otherwise. The same holds dually for ad^I and ad^I_c. Note that all action distances result in ∞ 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 $ad_c^O(Y,Z) = \infty$.

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* $\dagger \in \{I, O\}$, *we define*

$$td^\dagger(\sigma,\rho) = \begin{cases} \max_{1 \leq i \leq n} ad^\dagger(\alpha_i, \beta_i) & n = m \\ \infty & \text{otherwise.} \end{cases}$$

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,*

$$d(P,Q) = \sup_{\sigma \in tr(P)} \inf_{\rho \in tr(Q)} d(\sigma,\rho).$$

3. *The constrained trace distances,* $td_c^\dagger$ *and* td_c *, are defined like* $td^\dagger$ *and* td*, respectively, with* $ad_c^\dagger$ *taking the place of* $ad^\dagger$.

The trace distances which we will consider in this paper are td and td_c^O, where td_c^O does correspond to the first scenario described above, and td the second. We will let the variable $\mathcal{D}$ range over $\{td, td_c^O\}$, 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* $td^O(\sigma,\rho) \leq \varepsilon \wedge td^I(\sigma,\rho) \leq 0$ *iff* $td_c^O(\sigma,\rho) \leq \varepsilon$.

We define the set of states that can be reached from a starting state with a trace that lies within distance ε from a given trace σ. The definition is generic for $\mathcal{D} \in \{td, td_c^O\}$.

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

$$\underline{s \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma} = \{s' \mid \exists \rho \in A^* : s \xrightarrow{\rho} s' \wedge \mathcal{D}(\sigma,\rho) \leq \varepsilon\}.$$

For $S' \subseteq S$ *we set* $\underline{S' \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma} = \bigcup_{s \in S'} \underline{s \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma}$. *We define* $\underline{Q \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma} := \underline{S^0 \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma}$.

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 *ioconf*, *ioco* and the I/O refusal relation) can all be formulated in terms 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* $\mathcal{S}$ *as specification and a QTS* $\mathcal{I}$ *as implementation. We assume both* $\mathcal{I}, \mathcal{S}$ *being input-enabled. For* $0 \leq \varepsilon \leq 1$ *and* $\mathcal{D} \in \{td, td_c^O\}$, *we define* $\mathcal{I} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{S}$ *iff* $\mathcal{D}(\mathcal{I}, \mathcal{S}) \leq \varepsilon$.

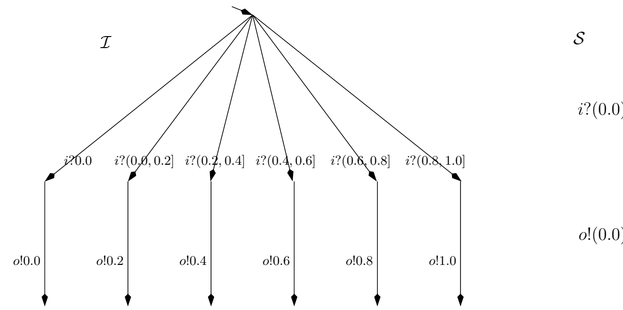

Figure 2: Example 5.4

Thus, we define $\mathcal{I} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{S}$ as $tr(\mathcal{I}) \subseteq_\varepsilon^\mathcal{D} tr(\mathcal{S})$, and we obtain by Lemma 4.2 that $\mathcal{I} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{S}$ iff $tr(\mathcal{I}) \subseteq B^\mathcal{D}(tr(\mathcal{S}), \varepsilon)$. If $\varepsilon = 0$, then $\sqsubseteq_\varepsilon^\mathcal{D}$ reduces to trace inclusion. Note that $\sqsubseteq_\varepsilon^\mathcal{D}$ for $\varepsilon \neq 0$ is not a preorder, since transitivity does not hold: from $\mathcal{P} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{R}$ we can not conclude that $\mathcal{P} \sqsubseteq_\varepsilon^\mathcal{D} \mathcal{R}$. However, the triangle inequality that holds for $\mathcal{D}$ allows us to conclude that $\mathcal{P} \sqsubseteq_{2\varepsilon}^\mathcal{D} \mathcal{R}$.

With the following lemma we get a different characterization of $\sqsubseteq_\varepsilon^\mathcal{D}$.

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

$$out(\underline{\mathcal{I} \ \mathbf{after}_0^\mathcal{D} \ \sigma}) \subseteq_\varepsilon^\mathcal{D} out(\underline{\mathcal{S} \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma})$$

5.2 $\mathbf{qioco}_\varepsilon^\mathcal{D}$

The formulation of $\sqsubseteq_\varepsilon^\mathcal{D}$ 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 $\mathcal{S}$ and $\mathcal{I}$, with $\mathcal{I}$ 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 (ε, to be exact), it is necessary to consider also traces that are at most ε 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 $B^\mathcal{D}(tr(\mathcal{S}), \varepsilon)$, i.e. to the traces that are at most ε off from the trace-set of the specification.

Definition 5.3 $\mathcal{I}$ $\mathbf{qioco}_\varepsilon^\mathcal{D}$ $\mathcal{S}$ *iff* $\forall \sigma \in B^\mathcal{D}(tr(\mathcal{S}), \varepsilon)$:

$$out(\underline{\mathcal{I} \ \mathbf{after}_0^\mathcal{D} \ \sigma}) \subseteq_\varepsilon^\mathcal{D} out(\underline{\mathcal{S} \ \mathbf{after}_\varepsilon^\mathcal{D} \ \sigma}).$$

Example 5.4 *In Figure 2, we see* $\mathcal{I}$*, the implementation, and* $\mathcal{S}$*, the specification*[3]*. From the starting state, we have outgoing transitions, all labeled with* $i?$. *After input* $i?(0.0)$, *specification* $\mathcal{S}$ *indicates that only output* $o!(0.0)$ *is correct:* $tr(\mathcal{S}) = \{i?(0.0) \cdot o!(0.0)\}$. *The implementation* $\mathcal{I}$ *yields after inputs* $i?(x)$ *with* $x \in (y - 0.2, y]$ *output* $o!(y)$, *for* $y = 0.2, 0.4, 0.6, 0.8, 1.0$.

[3]For the sake of simplicity we do not bother to make $\mathcal{I}$ input-complete and non-blocking on outputs.

We have $\mathcal{I}\,\mathbf{qioco}_\varepsilon^{td_c^O}\,\mathcal{S}$ for all $\varepsilon \in [0,1]$, because the only trace of length 1 in $B^{td_c^O}(tr(\mathcal{S}),\varepsilon)$ is $i?(0)$. We then have $out(\underline{\mathcal{I}\ \mathbf{after}_0^{\mathcal{D}}\ i?(0)}) = \{o!(0.0)\} = out(\underline{\mathcal{S}\ \mathbf{after}_0^{\mathcal{D}}\ i?(0)})$, i.e. the delivered output coincides exactly with the expected one. However, $\mathcal{I}\,\mathbf{qioco}_\varepsilon^{td}\,\mathcal{S}$ only for $\varepsilon \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The reason for this is that $B^{td}(tr(\mathcal{S}),\varepsilon)$ contains $\{i?(x) \mid x \in [0,\varepsilon]\}$, and for, e.g. $\varepsilon = 0.1$ and $i?(0.05) \in B^{td}(tr(\mathcal{S}), 0.1)$, $out(\underline{\mathcal{I}\ \mathbf{after}_0^{\mathcal{D}}\ i?(0.05)}) = \{o!(0.2)\}$. $td(o!(0.2), o!(0.0)) = 0.2 > \varepsilon$, 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 $\mathbf{qioco}^{\mathcal{D}}$ expressed as trace inclusion

It is a folklore result that *ioco*-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 $\mathcal{I}$ and $\mathcal{S}$ be input-enabled QTSs with the same action signature. Then $\mathcal{I}\ \mathbf{qioco}_\varepsilon^{\mathcal{D}}\ \mathcal{S}$ iff $\mathcal{D}(\mathcal{I},\mathcal{S}) \le \epsilon$.*

This result allows us to express $\mathbf{qioco}^{\mathcal{D}}$ 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 $\forall \sigma \in A^* : s, s' \in \underline{Q\ \mathbf{after}_0^{\mathcal{D}}\ \sigma}$ implies $\forall a \in A_I : s \xrightarrow{a}$ iff $s' \xrightarrow{a}$.*

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

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

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

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

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

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

Now we can characterize $\mathbf{qioco}_\varepsilon^{\mathcal{D}}$ in terms of trace inclusion.

Theorem 5.10 *Let $\mathcal{I}$ be an input-enabled QTS and $\mathcal{S}$ a well-formed one. Then*

$$\mathcal{I}\ \mathbf{qioco}_\varepsilon^{\mathcal{D}}\ \mathcal{S} \iff tr(\mathcal{I}) \subseteq tr(\Gamma(B_\varepsilon^{\mathcal{D}}(\mathcal{S}))).$$

5.4 The $\mathbf{qioco}^{\mathcal{D}}$ distance

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

Definition 5.11 ($d_{qioco}^{\mathcal{D}}$) *Let $\mathcal{I}$ be an input-enabled QTS and $\mathcal{S}$ a QTS. Then we define:*

$$d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S}) = \inf\{\varepsilon \in [0,1]_\infty \mid \mathcal{I}\ \mathbf{qioco}_\varepsilon^{\mathcal{D}}\ \mathcal{S}\}.$$

The following result extends Theorem 5.5 to the ioco-distance.

Theorem 5.12 *Let $\mathcal{I}$ and $\mathcal{S}$ be input-enabled QTSs with the same action signature. Then $d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S}) = \mathcal{D}(\mathcal{I},\mathcal{S})$.*

A different formulation of the above definition sheds light on how we can approximate $d_{qioco}^{\mathcal{D}}$ by means of *testing*. Another way to formulate $d_{qioco}^{\mathcal{D}}$ is as follows:

$$d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S}) = \sup\{\ \varepsilon\ \in [0,1]_\infty \mid \forall \varepsilon' < \varepsilon : \mathcal{I}\ \cancel{\mathbf{qioco}}_{\varepsilon'}^{\mathcal{D}}\ \mathcal{S}\}.$$

Using Lemma 5.10, this can be transformed to

$$\sup\{\ \varepsilon \in [0,1]_\infty \mid \forall \varepsilon' < \varepsilon : tr(\mathcal{I}) \cap \overline{tr(\Gamma(B_{\varepsilon'}^{\mathcal{D}}(\mathcal{S})))} \ne \emptyset\}.$$

Thus for all $\varepsilon < d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S})$, $tr(\mathcal{I}) \cap \overline{tr(\Gamma(B_\varepsilon^{\mathcal{D}}(\mathcal{S})))} \ne \emptyset$, i.e. $\exists \sigma \in tr(\mathcal{I})$ which is not element of $tr(\Gamma(B_\varepsilon^{\mathcal{D}}(\mathcal{S})))$. A testing approach to approximate $d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S})$ 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 $\sigma \in tr(\mathcal{I})$ with $\sigma \notin tr(\Gamma(B_\varepsilon^{\mathcal{D}}(\mathcal{S})))$ we can conclude that the chosen ε was too small. We must then derive an $\varepsilon' > \varepsilon$ from σ such that $\sigma \in tr(\Gamma(B_{\varepsilon'}^{\mathcal{D}}(\mathcal{S})))$. With this new ε' we start testing from the beginning and synthesize another trace σ', which gives us an ε'', and so on. In this way we approximate $d_{qioco}^{\mathcal{D}}(\mathcal{I},\mathcal{S})$. 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 $\mathbf{qioco}_\varepsilon^{\mathcal{D}}$ distance between an input-enabled QTS $\mathcal{I}$ and a QTS $\mathcal{S}$ 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 $\mathcal{I}$ be a QTS, input-enabled. A* trace function i of $\mathcal{I}$ *is a function $i : tr(\mathcal{I}) \rightarrow A_O$ with the property $i(\sigma) = \alpha!$ implies $\sigma \cdot \alpha! \in tr(\mathcal{I})$. The set of all trace functions of $\mathcal{I}$ is denoted as $TF(\mathcal{I})$.*

If $\sigma \in tr(\mathcal{I})$, and $i \in TF(\mathcal{I})$, then $i(\sigma) \in out(\underline{\mathcal{I}\ \mathbf{after}_0^{\mathcal{D}}\ \sigma})$. A trace function thus picks one output from several and thus resolves the nondeterminism in outputs of $\mathcal{I}$ after the execution of σ. Different executions of $\mathcal{I}$ are described by different trace functions. Since $\mathcal{I}$ 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 $\mathcal{S} = \langle S, S^0, L, \rightarrow\rangle$ be a QTS, $i \in TF(\mathcal{I})$ and $\mathcal{D} \in \{td, td_c^O\}$. We define for $\mathcal{S}$ and i a family of functions, $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma} : S \rightarrow [0,1]_\infty$ with $\sigma \in A^*, i(\sigma)\downarrow$ as follows. (1) $\mathsf{curr\_dist}^{\mathcal{D}}_{\lambda}(s) = 0$ if $s \in S^0$, and ∞ otherwise; (2) for $\alpha = i(\sigma)$ or $\alpha \in A_I$: $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot a}(s) = \inf_{s' \xrightarrow{b} s} \max\{\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}(s'), \mathcal{D}(a,b)\}$.*

Then $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}(s)$ is the minimal trace distance w.r.t. $\mathcal{D}$ of a trace σ from the set of traces $\{\rho \in A^* \mid \exists s_0 \in S^0 : s_0 \xrightarrow{\rho} s\}$, as is stated in Theorem 6.3.

Theorem 6.3 $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}(s) = \mathcal{D}(\sigma, \{\rho \mid \exists s_0 \in S^0 : s_0 \xrightarrow{\rho} s\})$.

Corollary 6.4 $\inf_{s\in S} \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}(s) = \mathcal{D}(\sigma, tr(\mathcal{S}))$.

For a more convenient construction of the $\mathsf{curr\_dist}$ functions in the algorithm to come, we introduce the operator $\mathcal{C} : (S \rightarrow [0,1]_\infty) \times A \times \{td, td_c^O\} \rightarrow (S \rightarrow [0,1]_\infty)$ as follows:

$$\mathcal{C}(c, \alpha, \mathcal{D}) = s \mapsto \inf_{s' \xrightarrow{\beta} s} \max\{c(s'), \mathcal{D}(\alpha, \beta)\}).$$

Clearly, $\mathcal{C}(\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}, \alpha, \mathcal{D}) = \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot\alpha}$.

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 the first to approximate $d^{\mathcal{D}}_{qioco}$. Again we assume that $\mathcal{I}$ is an input-enabled QTS representing the specification, and $\mathcal{S} = \langle S, S^0, L, \rightarrow\rangle$ is a QTS representing the specification.

The first algorithm is Algorithm 2. This depicts a nondeterministic procedure MQOTF, which takes five parameters, $i, \mathcal{S}, n, \mathcal{D}, \varepsilon$. $i \in TF(\mathcal{I})$ is a trace function representing 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. $\mathcal{D} \in \{td, td_c^O\}$ 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, σ), 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 $cd = \max\{\varepsilon, \mathcal{D}(\sigma, tr(\mathcal{S}))\}$. The main purpose of MQOTF is to construct the function $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}$ step-by-step, where σ is the trace synthesized during testing.
In lines 2–5, several local variables are initialized: σ is the trace observed so far, and is initialized with λ. cd keeps track of the lower bound of the distance of the observed trace to $tr(\mathcal{S})$ and is initialized with parameter ε. $curr\_dist$ is the current $\mathsf{curr\_dist}_{\sigma}$ function and is initialized with $\mathsf{curr\_dist}_{\lambda}$. 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 $\mathcal{D}(\sigma, \rho) \le cd$. 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 body of the **while**-loop is a nondeterministic algorithm: execution starts either on line 7 or 12. On line 7, an input $\alpha? \in A_I$ is chosen such

Algorithm 2 The distance measuring algorithm

Require: $\mathcal{S} = \langle S, S^0, L, \rightarrow\rangle$ is a QTS, i is a trace function of the IUT, $n \in \mathbb{N}$, $\mathcal{D} \in \{td, td_c^O\}$, $\varepsilon \in [0,1]$.
1: **procedure** MQOTF$(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$
2: $\quad \sigma \leftarrow \lambda$
3: $\quad cd \leftarrow \varepsilon$
4: $\quad curr\_dist = \mathsf{curr\_dist}^{\mathcal{D}}_{\lambda}$
5: $\quad M \leftarrow S^0$
6: $\quad$ **while** $cd < \infty \wedge |\sigma| \le n$ **do**
7: $\quad\quad [\ \alpha? \in A_I$ and $\underline{M\ \mathbf{after}^{\mathcal{D}}_{\varepsilon}\ \alpha? \ne \emptyset]} \rightarrow$
8: $\quad\quad\quad curr\_dist \leftarrow \mathcal{C}(curr\_dist, \alpha?, \mathcal{D})$
9: $\quad\quad\quad M \leftarrow \{s \mid curr\_dist(s) \le cd\}$
10: $\quad\quad\quad \sigma \leftarrow \sigma \cdot \alpha?$
11: $\quad\quad$ **end**
12: $\quad\quad$ [**true**] $\rightarrow\ \ \alpha! \leftarrow i(\sigma)$
13: $\quad\quad\quad curr\_dist \leftarrow \mathcal{C}(curr\_dist, \alpha!, \mathcal{D})$
14: $\quad\quad\quad cd \leftarrow \max\{cd, \inf_{s\in S} curr\_dist(s)\}$
15: $\quad\quad\quad M \leftarrow \{s \mid curr\_dist(s) \le cd\}$
16: $\quad\quad\quad \sigma \leftarrow \sigma \cdot \alpha!$
17: $\quad\quad$ **end**
18: $\quad$ **end while**
19: $\quad$ **return**(cd, σ)
20: **end procedure**

that $\underline{M\ \mathbf{after}^{\mathcal{D}}_{cd}\ \alpha?} \ne \emptyset$. If such an $\alpha?$ exists, $curr\_dist$ is updated, new menu M is defined, and $\alpha?$ is appended to σ (lines 8–10). Note that cd is not updated, since $\sigma\cdot\alpha?$ has the same trace distance to $tr(\mathcal{S})$ as σ. 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 $curr\_dist$, cd, M and σ. Note that cd is only increased if $\mathcal{D}(\sigma\cdot\alpha!, tr(\mathcal{S}))$ is larger than ε. Once the **while**-loop terminates, line 19 is reached. The computed distance cd, together with σ is then returned.

MQOTF returns (cd, σ), i.e. the trace distance of one trace only. Assuming that $cd \ge \mathcal{D}(\sigma, tr(\mathcal{S}))$ (this is shown in Section 6.3), MQOTF can be used to approximate $d^{\mathcal{D}}_{qioco}(\mathcal{I}, \mathcal{S})$, 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 $\mathcal{S}, \mathcal{I}$ and $\mathcal{D}$. 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. $\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}$, for $\mathcal{D} \in \{td, td_c^O\}$. Soundness means that, whenever $\mathcal{I}\,\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$ than for all $n \in \mathbb{N}$, $i \in TF(\mathcal{I})$ and possible return values (cd, σ) from MQOTF$(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$, $cd = \varepsilon$ holds. The algorithm is also complete, i.e. if $\mathcal{I}\,\not\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$, then there is a trace function $i \in TF(\mathcal{I})$ and a run procedure MQOTF$(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$ with return value (cd, σ) such that $cd > \varepsilon$.

Integral part of a soundness proof is to show that the fol-

Algorithm 3 Approximating $d^{\mathcal{D}}_{qioco}(\mathcal{I}, \mathcal{S})$

```
Require: S = ⟨S, S^0, L, →⟩ is a QTS, I an input-enabled
    QTS, n ∈ ℕ, D ∈ {td, td_c^O}.
n' ← 0
cd ← 0
σ ← λ
while n' ≤ n and cd < ∞ do
    [true] → let i ∈ TF(I), m ∈ ℕ in
        (cd, σ) ← MQOTF(i, S, m, D, cd)
        n' ← n' + 1
    end
end while
```

lowing property of Algorithm 2 holds: whenever execution reaches line 6 it holds: (1) $curr\_dist = \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}$; (2) $cd = \max\{\mathcal{D}(\sigma, tr(\mathcal{S})), \varepsilon\}$; (3) $M = \{s \mid curr\_dist(s) \leq cd\}$; (4) $|\sigma| \leq n + 1$.

These conditions are easily verified when line 6 is entered for the first time. Then $\sigma = \lambda, cd = \varepsilon$ ($\mathcal{D}(\lambda, tr(\mathcal{S})) = 0$), $curr\_dist = \mathsf{curr\_dist}^{\mathcal{D}}_{\lambda}, M = S^0 = \{s \mid curr\_dist(s) = 0\}$, and $|\sigma| = 0$. If we assume that all four condition hold and additionally $M \neq \emptyset$ 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 precondition of line 7 holds and the line is nondeterministically chosen, then action $\alpha? \in A_I$ is the input selected to be sent to the implementations (which is only implicitly done by appending $\alpha?$ to σ). In line 8, $curr\_dist$ is updated. From the definition of $\mathcal{C}$ it is easy to see that then $curr\_dist = \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot\alpha?}$ on line 9. Important to note is that in lines 8–10 the value of cd is not updated. The reason is that in fact $\inf_{s\in S} \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot\alpha?}(s) = \inf_{s\in S} \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}(s)$, since the input $\alpha?$ is chosen to not deviate more than $\varepsilon \leq cd$ from the specified inputs. The trace distance of $\sigma \cdot \alpha?$ to $\mathcal{S}$ is therefore equal to that of σ. When we return from line 10 to line 6, the four conditions are thus still satisfied.

If line 12 is chosen, output $\alpha!$ is received from the implementation (symbolized by consulting the trace function). In line 13, $curr\_dist$ is updated from $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}$ to $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot\alpha!}$. In line 13, cd is updated. By the precondition and Theorem 6.3, then $cd = \max\{\max\{\varepsilon, \mathcal{D}(\sigma, tr(\mathcal{S})), \mathcal{D}(\sigma\cdot\alpha!, tr(\mathcal{S}))\}\} = \max\{\varepsilon, \mathcal{D}(\sigma \cdot \alpha!, tr(\mathcal{S}))\}$. 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 MQOTF returns a result (cd, σ), then $cd = \max\{\varepsilon, \mathcal{D}(\sigma, tr(\mathcal{S}))\}$.

To prove now soundness, we assume that $\mathcal{I}\,\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$, but that a run of $\text{MQOTF}(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$ for $i \in TF(\mathcal{I})$ returns (cd, σ) with $cd > \varepsilon$. We know then that $cd = \mathcal{D}(\sigma, tr(\mathcal{S}))$. Then there is also a prefix $\sigma'\cdot\alpha!$ of σ such that $\mathcal{D}(\sigma', tr(\mathcal{S})) \leq \varepsilon$, but $\mathcal{D}(\sigma' \cdot \alpha!, tr(\mathcal{S})) > \varepsilon$ (only outputs can increase the distance of a trace to $tr(\mathcal{S})$). Then $\sigma' \in B^{\mathcal{D}}(tr(\mathcal{S}), \varepsilon)$, and $\alpha! \in out(\underline{\mathcal{I}\ \mathbf{after}^{\mathcal{D}}_0\ \sigma'})$. However, this implies also that $out(\underline{\mathcal{I}\ \mathbf{after}^{\mathcal{D}}_0\ \sigma'}) \not\sqsubseteq^{\mathcal{D}}_{\varepsilon} out(\underline{\mathcal{S}\ \mathbf{after}^{\mathcal{D}}_0\ \sigma'})$, a contradiction to the assumption $\mathcal{I}\,\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$.

To show completeness we have to prove that, if $\mathcal{I}\,\mathbf{qio\not{c}o}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$, then there is a trace function $i \in TF(\mathcal{I})$ and a run of procedure $\text{MQOTF}(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$ with return value (cd, σ) such that $cd > \varepsilon$. $\mathcal{I}\,\mathbf{qio\not{c}o}^{\mathcal{D}}_{\varepsilon}\,\mathcal{S}$ implies according to the definition of $\mathbf{qioco}^{\mathcal{D}}_{\varepsilon}$ that there is a $\sigma \in B^{\mathcal{D}}(tr(\mathcal{S}), \varepsilon)$ with $out(\underline{\mathcal{I}\ \mathbf{after}^{\mathcal{D}}_0\ \sigma}) \not\sqsubseteq^{\mathcal{D}}_{\varepsilon} out(\underline{\mathcal{S}\ \mathbf{after}^{\mathcal{D}}_{\varepsilon}\ \sigma})$. There is thus an output $\alpha! \in out(\underline{\mathcal{I}\ \mathbf{after}^{\mathcal{D}}_0\ \sigma})$ with $\mathcal{D}(\{\alpha!\}, out(\underline{\mathcal{S}\ \mathbf{after}^{\mathcal{D}}_{\varepsilon}\ \sigma})) > \varepsilon$, and moreover, $\mathcal{D}(\sigma\cdot\alpha!, tr(\mathcal{S})) > \varepsilon$. This implies that $\{\sigma, \sigma\cdot\alpha!\} \subseteq tr(\mathcal{I})$, i.e. there is also a trace function $i \in TF(\mathcal{I})$ with $i(\sigma) = \alpha!$. Let $n = |\sigma|$. Since $\sigma \in B^{\mathcal{D}}(tr(\mathcal{S}), \varepsilon)$, we can assume that there is a run through $\text{MQOTF}(i, \mathcal{S}, n, \mathcal{D}, \varepsilon)$ such that we enter line 7 of Algorithm 2 with the following conditions fulfilled: (1) $curr\_dist = \mathsf{curr\_dist}^{\mathcal{D}}_{\sigma}$; (2) $cd = \varepsilon \geq \mathcal{D}(\sigma, tr(\mathcal{S}))$; (3) $M = \{s \mid curr\_dist(s) \leq \varepsilon\}$; (4) $n' = n$. If the algorithm proceeds then to line 12, trace function i will return output $\alpha!$, $curr\_dist$ will be updated to $\mathsf{curr\_dist}^{\mathcal{D}}_{\sigma\cdot\alpha!}$ and cd to $\max\{\varepsilon, \inf_{s\in S} curr\_dist(s)\} == \mathcal{D}(\sigma \cdot \alpha!, tr(\mathcal{S}))$. Thus $cd > \varepsilon$. Since n' will be updated to $n+1$, the algorithm will terminate and return with $(cd, \sigma\cdot o)$, where $cd > \varepsilon$. 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 $\mathcal{D} = td$. Since d^{td}_{qioco} 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 σ, the tester decides to stimulate the IUT with an action $\alpha?$, then the new test history becomes $\sigma\alpha?$; in case of an observation, the test accounts for all possible continuations $\sigma\beta!$ with $\beta! \in L^O$ 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 $\mathcal{S}_t$, which in each state either selects one input action, or enables all output actions.

Definition 7.1 *A test case (or test) t for $\mathcal{S}$ is a prefix-closed subset of A^* such that, (1) if $\sigma\alpha? \in t$, then $\sigma\beta \notin t$ for any $\beta \in A$ with $\alpha? \neq \beta$, (2) if $\sigma\alpha! \in t$, then $\sigma\beta! \in t$ for all $\beta! \in A_O$, (3) if $\sigma \notin tr(\mathcal{S})$, then $\mathrm{last}(\sigma) \in A_O$ and σ is no proper prefix of any $\sigma' \in t$, and (4) t does not contain any strictly increasing chain $\sigma_0 \prec \sigma_1 \prec \sigma_2 \prec \ldots$.*

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 $\mathcal{S}$ by $TESTS(\mathcal{S})$.

The following lemma states that every behavior of the specification $\mathcal{S}$ can be tested.

Lemma 7.2 *For all $\sigma \in tr(\mathcal{S})$, there is a test $t \in TESTS(\mathcal{S})$ such that $\sigma \in t$.*

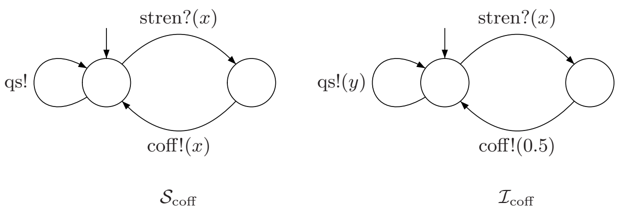

Figure 3: Coffee Machine Specification $\mathcal{S}_{\text{coff}}$ and Implementation $\mathcal{I}_{\text{coff}}$

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 $\mathcal{S}_t$.

Definition 7.3 *Let t be a test for QTS $\mathcal{S}$. The QTS-representation of t is the QTS $\mathcal{S}_t = \langle S_t, S_t^0, L_t, \rightarrow_t\rangle$ given as follows. The states are all traces in t, i.e. $S_t = t$; the initial state is the empty trace, i.e. $S_t^0 = \{\lambda\}$; its labels are exactly the labels of $\mathcal{S}$, i.e. $L_t = L$; and the transition relation $\rightarrow_t \subseteq S_t \times A_t \times S_t$ is given by $\{(\sigma, \alpha, \sigma\alpha) \mid \sigma\alpha \in t\}$.*

It immediately follows that $tr(\mathcal{S}_t) = t$.

Example 7.4 *Figure 3 shows the specification $\mathcal{S}_{\text{coff}}$ 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 $x \in [0,1]$ and that* stren?(x) *is followed by a* coff!(x)*. In reality there are thus uncountably many states and transitions. To make the QTS output-complete, we add a label* qs!*, which represents quiescence, i.e. absence of outputs. The set $t = \{$stren?(0.8)coff!(x)) $\mid x \in [0,1]\} \cup \{$stren?(0.8)qs!($y$)) $\mid y \in [0,1]\}$. The verdict of a trace* stren?(0.8)coff!(x) *is given by $v($stren?(0.8)coff!(x)$) = |0.8-x|$; the verdict of trace* stren?(0.8)qs!(x) *is $v($stren?(0.8) coff!(x)$) = \infty$.*

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

Definition 7.5 *Let t be a test for QTS $\mathcal{S}$. The* quantitative verdict function *$v_{\mathcal{S}}$ for $\mathcal{S}$ is the function $v_{\mathcal{S}} : leaves(t) \rightarrow [0,1]_\infty$, with $v_{\mathcal{S}}(\sigma) = td(\sigma, tr(\mathcal{S}))$. We call the pair $\mathbf{qt} = (t, v_{\mathcal{S}})$ a* evaluated test *for $\mathcal{S}$, and $ET(\mathcal{S})$ 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 σ with $td(\sigma, \mathcal{S}) = \delta \leq 1$, there exists an evaluated test $(t, v) \in ET(\mathcal{S})$ 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 $a?(0.50)$, but in practice, stimulus $a?(0.51)$ occurs. Similarly, the IUT may produce an output $b!(0.30)$, but due to measurement imprecisions, we read it off as $b!(0.29)$. Thus, when we execute t on I with imprecision δ, an action a in t may synchronize with any action b in I within action distance δ.

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

Definition 7.7 *For two QTSs Q and P be two QTSs with the same action signatures. Let $\delta \geq 0$. We define the parallel composition with tolerance δ, denoted $Q\|_\delta P$, as the QTSs $\langle S, S^0, L, \rightarrow\rangle$ given by $S_{Q\|_\delta P} = S_Q \times S_P$ and $S^0_{Q\|_\delta P} = S^0_Q \times S^0_P$, and*

- $L_{Q\|_\delta P} = (L^I_Q, L^O_Q)$,
- $\rightarrow_{Q\|_\delta P} = \{(s,u) \xrightarrow{\alpha}_{Q\|_\delta P} (s', u') \mid s \xrightarrow{\alpha}_Q s' \wedge u \xrightarrow{\alpha}{}^{\delta}_P u'\}$.

Here, $\rightarrow^\delta_P$ denotes the δ-transition relation given by $s \xrightarrow{\alpha}{}^\delta_P s'$ iff there exists an $\alpha \in A_P$ with $ad(\alpha, \beta) \leq \delta$ and $s \xrightarrow{\beta}_P s'$.

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

Suppose we run, with an imprecision of at most δ, a test case t on implementation I. Then the set of all possible executions are exactly the traces of $\mathcal{S}_t\|_\delta I$.

Definition 7.8 *Let $\mathbf{t} = (t, v) \in ET(\mathcal{S})$ be an evaluated test for $\mathcal{S}$ and $T \subseteq ET(\mathcal{S})$ be a evaluated test suite for $\mathcal{S}$. Let $\delta > 0$. The set of* test executions *is given by $exec^\delta(\mathbf{t}, I) = tr(\mathcal{S}_t\|_\delta I)$.*

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 $\mathbf{t} = (t, v) \in ET(\mathcal{S})$ be an evaluated test for $\mathcal{S}$ and $T \subseteq ET(\mathcal{S})$ be a evaluated test suite for $\mathcal{S}$. Let $\delta \geq 0$. The* verdict *of $v^\delta_{\mathbf{t}}(I)$ is given by $v^\delta_{\mathbf{t}}(I) = \sup_{\sigma \in exec^\delta(t,I)} v_t(\sigma)$, and the* verdict *of $v_T(I)$ is given by $v^\delta_T(I) = \sup_{t \in T} v^\delta_t(I)$.*

Example 7.10 *Figure 3 depicts an implementation $\mathcal{I}_{\text{coff}}$ of a coffee machine, where the user always gets a coffee of strength 0.5. Note that $td(\mathcal{I}_{\text{coff}}, \mathcal{S}_{\text{coff}}) = 0.5$. However, if we run the $\mathcal{I}_{\text{coff}}$ against test t, then we obtain for $\delta = 0.1$ that $exec(t, \mathcal{I}_{\text{coff}}) = \{$stren?($y$)coff!($x$) $\mid y \in [0.7, 0.9], x \in [0.5, 0.6]\}$. Thus, $v^\delta_t(\mathcal{I}_{\text{coff}}) = 0.4$, which is witnessed by the trace* stren?(0.9)$\cdot$ coff!(0.5)*.*

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

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

1. *$exec(\mathbf{t}, I) = leaves(t) \cap B_\delta(tr(I))$.*
2. *$v^\delta_{\mathbf{t}}(I) = td(t\|_\delta I, \mathcal{S})$.*

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 $\mathcal{S}$ any conforming implementation passes all tests derived from $\mathcal{S}$ (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 worse case verdict that we obtain when we run all tests from $TESTS(\mathcal{S})$ against an implementation I is exactly the trace distance, corrected with the imprecision δ. Let $\gamma = td(I, \mathcal{S})$.

Theorem 7.12 (soundness & completeness)

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

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

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

Theorem 7.14 *Let $\mathcal{I}, \mathcal{S}$ be input-enabled QTSs with the same action signature. Then*

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

Here i ranges over the trace functions of $\mathcal{I}_\delta$ (c.f. 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 UIT 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 *curr_dist* and efficient methods to compute the function $\underline{\cdot\ \textbf{after}^{\mathcal{D}}\ \cdot}$.

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

[1] H. Bohnenkamp and A. Belinfante. Timed testing with TorX. In *Proc. FME 2005*, volume 3582 of *LNCS*, pages 173–188. Springer-Verlag, 2005.

[2] H. Bohnenkamp and M. Stoelinga. Quantitative testing. Technical Report AIB-2008-02, RWTH Aachen, Jan. 2008.

[3] L. B. Briones and H. Brinksma. A test generation framework for quiescent real-time systems. In *Proc. FATES '04*, volume 3395 of *LNCS*, pages 64–78. Springer-Verlag, 2005.

[4] C. Daws and P. Kordy. Symbolic robustness analysis of timed automata. In *FORMATS*, volume 4202 of *LNCS*, pages 143–155. Springer-Verlag, 2006.

[5] L. de Alfaro, M. Faella, and M. Stoelinga. Linear and branching metrics for quantatative transition systems. In *Proc. ICALP'04*, volume 3142 of *LNCS*, pages 97–109. Springer–Verlag, 2004.

[6] J.-C. Fernandez, C. Jard, T. Jeron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *Proc. CAV '96*, volume 1102 of *LNCS*, pages 348–359. Springer-Verlag, 1996.

[7] L. Frantzen, J. Tretmans, and T. Willemse. Test generation based on symbolic specifications. In *FATES 2004*, number 3395 in LNCS, pages 1–15. Springer-Verlag, 2005.

[8] L. Frantzen, J. Tretmans, and T. Willemse. A symbolic framework for model-based testing. In *FATES/RV 2006*, number 4262 in LNCS, pages 40–54. Springer-Verlag, 2006.

[9] V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *HART '97: Proceedings of the International Workshop on Hybrid and Real-Time Systems*, pages 331–345. Springer-Verlag, 1997.

[10] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In S. Graf and L. Mounier, editors, *Proc. 11th Int. SPIN Workshop (SPIN 2004)*, volume 2989 of *LNCS*, pages 109–126. Springer-Verlag, 2004.

[11] M. Krichen and S. Tripakis. An expressive and implementable formal framework for testing real-time systems. In *Proc. TESTCOM'05*, number 3502 in LNCS. Springer-Verlag, 2005.

[12] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In E. Brinksma, W. Grieskamp, and J. Tretmans, editors, *Proceedings of FATES'04*, volume 3395 of *LNCS*, pages 79–94, 2005.

[13] A. Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

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

[15] J. Tretmans and H. Brinksma. Torx: Automated model based testing. In A. Hartman and K. Dussa-Ziegler, editors, *Proc. 1st European Conf. on Model-Driven Software Engineering*, Nürnberg, 2003.

[16] M. v. d. Bijl, A. Rensink, and J. Tretmans. Compositional testing with ioco. In *Proc FATES '03*, volume 2931 of *LNCS*, pages 89–103. Springer-Verlag, 2004.

[17] M. v. Osch. Hybrid input-output conformance and test generation. In *Proc. FATES/RV '06*, volume 4262 of *LNCS*, pages 70–84. Springer-Verlag, 2006.