Formal methods in conformance testing : a probabilistic refinement

Lex Heerink and Jan Tretmans
Tele-Informatics and Open Systems group, Dept. of Computer Science
University of Twente, 7500 AE Enschede, The Netherlands
{heerink,tretmans}@cs.utwente.nl

Abstract

This paper refines the framework of 'Formal Methods in Conformance Testing' by introducing probabilities for concepts which have a stochastic nature. Test execution is refined into test runs, where each test run is considered as a stochastic process that returns a possible observation with a certain probability. This implies that not every possible observation that could be made, will actually be made. The development process of an implementation from a specification is also viewed as a stochastic process that may result in a specific implementation with a certain probability. Together with a weight assignment on implementations this introduces a valuation measure on implementations. The test run probabilities and the valuation measures are integrated in generalized definitions of soundness and exhaustiveness, which can be used to compare test suites with respect to their ability to accept correct, and to reject erroneous implementations.

Keywords

Conformance testing, test framework, test selection, formal methods, probabilities.

1 INTRODUCTION

Conformance testing is a way to assess the correctness of an implementation with respect to its specification by means of performing experiments on the implementation and observing its responses. In case the specification is given as a formal description we need formal definitions of testing concepts, such as correctness of an implementation with respect to a formal specification, a test purpose, a sound test case, test execution, test generation, etc. Currently, the standardization group on 'Formal Methods in Conformance Testing' (ISO/IEC JTC 1/SC 21/WG 7 project 54, ITU-T SG 10/Q 8) develops a framework for conformance testing based on formal methods defining these concepts [ISO96]. The framework defines terminology, abstract concepts, and minimal requirements on, and relations between these concepts. Since it is defined at a high level of abstraction, e.g., it abstracts from specific test generation algorithms, even from a specific formal description technique, use of the framework requires instantiating these concepts with specific choices for test generation algorithms, for the formal description technique, etc.

This paper builds on the framework of 'Formal Methods in Conformance Testing'. Its goal is to refine this framework by adding probabilities for those concepts which have a stochastic nature. The refinement concerns the testing process; probabilistic extensions of specification languages or models are not considered. The first addition is to consider test execution in a probabilistic setting. In the framework, test execution of a test case against an implementation under test is assumed to yield a unique observation. This assumption is replaced by refining test execution into a number of test runs. Each test run yields an observation of the implementation, but a number of test runs does not necessarily yield all possible observations [LS89]. A probability distribution is added to express which observations are likely to be obtained.

A second refinement of the framework concerns the extension of the soundness and the exhaustiveness of a test suite. Soundness refers to the property of a test suite to accept all conforming implementations, and exhaustiveness indicates that all nonconforming implementations are rejected [ISO96]. These predicates are generalized to measures in the vein of [BTV91, Bri93], which take into account the probability of the occurrence of implementations, the gravity of errors in implementations, and the above mentioned probability on observations made during test runs. It is indicated how these soundness and exhaustiveness measures can then be used to compare test suites, in order to select a good, or the best one.

The outline of this paper is as follows. In section 2 an overview of the framework 'Formal Methods in Conformance Testing' is given, as far as it is relevant for this paper, and some of the assumptions underlying this framework are discussed. Section 3 refines test execution into test runs, and it adds probabilities to the observations. In section 4 a valuation on implementations is defined, which assigns a value based on their probability of occurrence and on their weight. Section 5 uses the probability on test-run observations and the valuation on implementations to define soundness and exhaustiveness as two measures on test suites. Comparison of test suites based on these measures is briefly discussed. In section 6 the concepts are illustrated for labelled transition systems with inputs and outputs; this section may be read in parallel with the sections 2 till 5. Section 7 presents concluding remarks and items for further work.

2 FORMAL METHODS IN CONFORMANCE TESTING

The emerging international standard 'Formal Methods in Conformance Testing' defines a framework for the use of formal methods in conformance testing [ISO96]. It is intended to guide the testing process of an implementation with respect to a formal specification. In this section the main concepts of [ISO96], such as conformance, testing, and conformance testing, are presented, as far as they are needed for the subsequent sections. They are followed by a discussion of some of the explicit and implicit assumptions on which the framework is based.

Conformance The definition of conformance concerns implementations under test (IUT) and specifications, so a universe of implementations $IMPS$, and a universe of formal specifications $SPECS$ are assumed. Implementations are concrete, informal objects, such as pieces of hardware, or pieces of software. In order to reason formally about them, it is assumed that each implementation $IUT \in IMPS$ can be modelled by a formal object i_{IUT} in a formalism $MODS$, which is referred to as the universe of models. This hypothesis is referred to as the test assumption. Note that a model i_{IUT} is only assumed to exist; it is not known apriori.

Conformance is expressed by means of an implementation relation $\mathbf{imp} \subseteq MODS \times SPECS$. Implementation $IUT \in IMPS$ is considered **imp**-correct with respect to $s \in SPECS$ if the model $i_{IUT} \in MODS$ of IUT is **imp**-related to s: $i_{IUT}\ \mathbf{imp}\ s$. We use $I_s =_{\text{def}} \{i \in MODS \mid i\ \mathbf{imp}\ s\}$ for the set of **imp**-correct implementations, and $\overline{I_s} =_{\text{def}}\ MODS \backslash I_s$ for its complement.

Testing The behaviour of concrete implementations is investigated by performing experiments on the implementations and observing the reactions that the implementations produce to these experiments. Such experiments are called tests, and they are formally specified as elements of a universe of test cases TESTS. A set of test cases is called a test suite. The process of running a test against a concrete implementation is called test execution. Test execution leads to an observation in a domain of observations OBS. To each observation a verdict is assigned by a verdict assignment function $\mathit{verd}_t : \mathit{OBS} \to \{\mathbf{pass}, \mathbf{fail}\}$. It is said that a concrete implementation $\mathit{IUT} \in \mathit{IMPS}$ **passes** a test suite $T \subseteq \mathit{TESTS}$ if the test execution of all its test cases lead to an observation with verdict **pass**:

$$\begin{array}{ll} \mathit{IUT}\ \mathbf{passes}\ T =_{\text{def}} & \forall t \in T :\ \mathit{IUT}\ \mathbf{passes}\ t \\ \mathit{IUT}\ \mathbf{passes}\ t =_{\text{def}} & \text{test execution of } t \text{ against } \mathit{IUT} \text{ gives } \sigma \in \mathit{OBS}, \\ & \text{such that } \mathit{verd}_t(\sigma) = \mathbf{pass} \end{array} \tag{1}$$

An implementation fails test suite T if it does not pass: $\mathit{IUT}\ \mathbf{fails}\ T =_{\text{def}}\ \neg(\mathit{IUT}\ \mathbf{passes}\ T)$.

The interpretation of test execution, i.e., of IUT **passes** T, is given by modelling the process of test execution on models of implementations. By comparing the concrete observations made of IUT with the calculated observations of the model of test execution, conclusions can be drawn about the model i_{IUT}, in particular, a set of candidate models for i_{IUT} can be calculated. Let test execution be modelled as a function $\mathit{exec} : \mathit{TESTS} \times \mathit{MODS} \to \mathit{OBS}$, such that for each test case $t \in \mathit{TESTS}$ and each model $i \in \mathit{MODS}$, $\mathit{exec}(t, i)$ calculates the observation in OBS that results from executing t with the model i. If exec indeed faithfully models concrete test execution, then it can be concluded from successful test execution, IUT **passes** t, that the model of IUT is in the subset P_t of models for which a **pass**-verdict is calculated:

$$\begin{array}{ll} \text{let} & P_t =_{\text{def}}\ \{\ i \in \mathit{MODS} \mid \mathit{verd}_t(\mathit{exec}(t,i)) = \mathbf{pass}\ \} \\ \text{then} & \mathit{IUT}\ \mathbf{passes}\ t \iff i_{\mathit{IUT}} \in P_t \end{array} \tag{2}$$

and moreover for a test suite T:

$$\begin{array}{ll} \text{let} & P_T =_{\text{def}}\ \bigcap_{t \in T} P_t \\ \text{then} & \mathit{IUT}\ \mathbf{passes}\ T \iff i_{\mathit{IUT}} \in P_T \end{array} \tag{3}$$

In this way, for each test suite T, the universe of models of implementations MODS is partitioned into models in P_T, which model passing implementations, and models not in P_T. The set P_T is called the formal test purpose of T.

Conformance testing In order to judge whether a concrete implementation IUT conforms to its specification $s \in \mathit{SPECS}$ by means of testing, the notion of conformance, i.e., the set I_s, and test execution, i.e., the set P_T (3), have to be linked, so that from test execution an indication can be obtained whether $i_{\mathit{IUT}} \in I_s$, i.e., whether IUT conforms. A test suite is complete if it can distinguish exactly between all conforming and non-conforming implementations: $I_s = P_T$. Unfortunately, this is a very strong requirement for practical testing: complete test suites are usually infinite, and consequently not practically executable. Hence, [ISO96] poses a weaker requirement on test suites: they shall be sound, which means that at least all correct implementations (and possibly some incorrect implementations) will pass them:

$$\forall i \in \mathit{MODS} :\ i\ \mathbf{imp}\ s \Longrightarrow i \in P_T \tag{4}$$

In case all incorrect implementations (and possibly some correct ones) do not pass the execution of test suite T, the test suite is called exhaustive:

$$\forall i \in \mathit{MODS} :\ i\ \mathbf{imp}\ s \Longleftarrow i \in P_T \tag{5}$$

To quantify the error-detecting capability of a sound test suite a coverage measure can be defined, which expresses the extent to which a sound test suite is exhaustive ($\mathcal{P}(\mathit{TESTS})$ is the powerset of TESTS, i.e., the set of sets of test cases, so the set of test suites):

$$\mathit{cov} : \mathcal{P}(\mathit{TESTS}) \longrightarrow [0,1] \quad \text{satisfying} \quad P_{T_1} \supseteq P_{T_2} \Longrightarrow \mathit{cov}(T_1) \leq \mathit{cov}(T_2) \tag{6}$$

Assumptions In the definition of the formal testing framework of [ISO96] some explicit and implicit assumptions are made. The first assumption, which is explicitly stated, is the test assumption: any concrete implementation IUT can be modelled by an $i_{\mathit{IUT}} \in \mathit{MODS}$. In order to do any formal reasoning about implementations the existence of such a model should be assumed. However, the test assumption is unclear about unicity of i_{IUT}, i.e., about fully-abstractness with respect to observable behaviour, and about consequences of non-unicity.

A second assumption has to do with practical test execution. Execution of one single test case usually consists of several test runs, where each test run consists of applying the test case once to the implementation under test. Due to nondeterminism in the implementation under test each test run may lead to a different outcome, and the outcomes of several, independent test runs make up one observation. Let the class of possible test-run outcomes be $\mathcal{O}$, then an observation is a set of outcomes: $\mathit{OBS} = \mathcal{P}(\mathcal{O})$. Now $\mathit{exec} : \mathit{TESTS} \times \mathit{MODS} \to \mathcal{P}(\mathcal{O})$ calculates all possible test outcomes of given t and i. However, in the concrete test execution of t against an IUT it is difficult, or impossible, to be sure that all possible outcomes have really been obtained. Concrete test execution consists of performing a finite number of test runs, during which certain nondeterministic behaviours of the implementation may never be encountered. After any finite number of test runs it cannot be known whether all possible outcomes have been obtained or not. Consequently, if concrete test execution gives us an observation $O \subseteq \mathcal{O}$, we cannot conclude that $O = \mathit{exec}(t, i_{\mathit{IUT}})$ as above, and in [ISO96], but only that $O \subseteq \mathit{exec}(t, i_{\mathit{IUT}})$.

But even the conclusion $O \subseteq \mathit{exec}(t, i_{\mathit{IUT}})$ is not always valid; it depends on another assumption, viz. that the concrete observations obtained from test runs are always those which can be calculated from the model of test execution. This only holds if we assume that test cases are correctly implemented, i.e., that each test case is a valid model of its own implementation, and that *exec* correctly models the concrete observations that can be made during test runs. If the first assumption cannot be assumed to hold then we should verify or test the implementations of our test cases, for which we would have to derive and implement test cases again, which should also be tested, etc. Usually this assumption can be made since test cases are assumed to be an order of magnitude simpler than the implementations that are the aim of our testing. The second assumption requires an accurate modelling of the concrete test execution process, for any test case and any IUT, by means of the function *exec*, which is not always easy. Consider as an example the observation of a time-out. Time-outs are used to detect that an implementation does not react to a given stimulus. However, if no real-time requirements are specified for the reaction, then the time-out value should theoretically be infinite, which is practically infeasible. So it might be that a time-out is observed where only the implementation under test is slow. The observation of this time-out will usually not be modelled in *exec*, where the theoretically infinite timer is considered.

In the probabilistic additions to the formal framework in the next sections we will not challenge the test assumption, but we will reconsider the assumption that all test-run outcomes can be obtained during test execution: we decompose test execution into test runs, taking into account that during concrete test execution some outcomes might be missed; this will be discussed in section 3. Implemented test cases and test execution are, without change, assumed to be correctly modelled by test cases and a function *exec*, respectively.

3 TEST EXECUTION AS PROBABILISTIC TEST RUNS

It was argued in section 2 that the application of a single test case to a concrete implementation usually consists of multiple test runs, where the outcome of each test run may be different, due to nondeterminism in the implementation itself (e.g., a gambling machine modelling the tossing of a coin), or due to nondeterminism introduced by interaction of the implementation with its environment (e.g., a fileserver writing a file to disk that is nondeterministically interrupted by the operating system when the disk is full). Each time a test-run experiment is repeated, it may result in another outcome, and it is never known when all possible outcomes of the *IUT* have been obtained. Consequently, test execution results in a set of test-run outcomes, where some outcomes might be more likely to occur than others. This leads to considering the occurrence of an outcome as a stochastic experiment, in which a single outcome from the set of possible outcomes is drawn. The probability of an outcome to occur can be thought of as depending on the frequency with which the implementation resolves the nondeterministic choices leading to the different outcomes.

Let $\mathcal{O}$ be the class of outcomes, and let test execution be correctly modelled by $exec : TESTS \times MODS \to \mathcal{P}(\mathcal{O})$ (cf. section 2), then in each test run of test case t against implementation *IUT*, modelled by i, an outcome σ from sample set $exec(t, i)$ is drawn, where each outcome in $exec(t, i)$ has a nonzero probability to occur. This can be described by viewing a test-run as a stochastic experiment that produces an outcome $\underline{\sigma}(t, i)$ that takes its value in $exec(t, i)$. Consequently, for every t and i a probability distribution $P_o^{t,i} : \mathcal{P}(exec(t, i)) \to [0, 1]$ is assumed, indicating the probability that testing implementation i with test t leads to an outcome in $O \subseteq exec(t, i)$, viz.

$$P_o^{t,i}(O) =_{\text{def}} \ Pr\{\underline{\sigma}(t, i) \in O\} \tag{7}$$

Note that the distribution of the random variable $\underline{\sigma}(t, i)$ depends on the implementation i and test t; for each i and t, $\underline{\sigma}(t, i)$ may have another distribution.

Instead of combining outcomes into one observation $O \subseteq \mathcal{O}$, and then assigning the verdict $verd_t(O)$, we will assign verdicts to the outcomes, and then combine these outcome-verdicts into one verdict for the observation. The verdict for the observation will be **pass** if all outcome-verdicts are **pass**. Note that by combining outcome-verdicts into an observation-verdict instead of combining outcomes into an observation, we loose some observation power, e.g., consider a system which shall produce either always x or always y. Two test runs yield the outcomes x and y, respectively. Since they are both valid outcomes the test-run verdicts assigned will be both **pass**, whereas the verdict assigned to the observation $\{x, y\}$ would be **fail**.

Let $v_t : \mathcal{O} \to \{\textbf{pass}, \textbf{fail}\}$ be a verdict assignment to outcomes, then the probability measure $P_o^{t,i}$ can be used to induce a probability measure on sets of verdicts in $\{\textbf{pass}, \textbf{fail}\}$. As each outcome $\sigma \in exec(t, i)$ uniquely leads to a verdict $v_t(\sigma)$ this holds in particular for the outcome $\underline{\sigma}(t, i)$. As $\underline{\sigma}(t, i)$ is stochastically determined, it follows that the verdict assignment v_t is a stochastic function, and hence a probability distribution $P_v^{t,i} : \mathcal{P}(\{\textbf{pass}, \textbf{fail}\}) \to [0, 1]$ is obtained. The probability that a test run of t with i results in verdict **pass** is the cumulative probability that an outcome in $exec(t, i)$ occurs that leads to a **pass**-verdict:

$$P_v^{t,i}(\{\textbf{pass}\}) =_{\text{def}} \ Pr\{v_t(\underline{\sigma}(t, i)) = \textbf{pass}\} = P_o^{t,i}(\{\sigma \mid v_t(\sigma) = \textbf{pass}, \sigma \in exec(t, i)\}) \tag{8}$$

A straightforward extension of the measure $P_v^{t,i}$ on test-run verdicts to test-case and test-suite execution verdicts can be given. To integrate test-case and test-suite execution we consider a test suite as a multi-set of test cases rather than a set as in section 2. Multiple test runs of a single test case t can then be represented by multiple occurrences of that test case in the multi-set, written as t^n, so a test suite specifies the test cases it contains together with the number of test runs for each test case. For test suite $\langle t_1, \ldots, t_n \rangle$ we have the random variables $\langle \underline{\sigma_1}(t_1, i), \ldots, \underline{\sigma_n}(t_n, i) \rangle$, where each $\underline{\sigma_k}(t_k, i)$ $(1 \leq k \leq n)$ takes its value in $exec(t_k, i)$ according to the probability distribution $P_o^{t_k,i}$. According to (1,3) a test suite is passed by i if all the test cases in the test suite are passed. Analogously, the probability that implementation i passes test suite $T = \langle t_1, \ldots, t_n \rangle$ equals the probability that all tests $t_1, \ldots, t_n$ are passed by i:

$$P_v^{T,i}(\{\mathbf{pass}\}) =_{\mathrm{def}} \; Pr\{v_{t_1}(\underline{\sigma_1}(t_1, i)) = \mathbf{pass}, \ldots, v_{t_n}(\underline{\sigma_n}(t_n, i)) = \mathbf{pass}\} \tag{9}$$

Under the assumption that the random variables $\underline{\sigma_1}(t_1, i), \ldots, \underline{\sigma_n}(t_n, i)$ are independent, that is, the outcome of a test run does not depend on previous (or future) test runs, it follows that

$$P_v^{T,i}(\{\mathbf{pass}\}) = \prod_{k=1}^{n} Pr\{v_{t_k}(\underline{\sigma_k}(t_k, i)) = \mathbf{pass}\} = \prod_{k=1}^{n} P_v^{t_k,i}(\{\mathbf{pass}\}) \tag{10}$$

The probability to fail T immediately follows: $P_v^{T,i}(\{\mathbf{fail}\}) =_{\mathrm{def}} 1 - P_v^{T,i}(\{\mathbf{pass}\})$. It is evident that $P_v^{T,i}$ denotes a probability measure on $\mathcal{P}(\{\mathbf{pass}, \mathbf{fail}\})$, which has the obvious property, expressed in the next proposition, that for non-zero probability of **fail** in a test run, test case execution will finally result in **fail** if enough test runs are performed.

Proposition 1 If $P_v^{t,i}(\{\mathbf{pass}\}) < 1$ then $\lim_{n\to\infty} P_v^{t^n,i}(\{\mathbf{pass}\}) = 0$ $\square$

4 A VALUATION ON IMPLEMENTATIONS

By considering test runs as stochastic experiments, section 3 formalized the notion of passing a test suite in the form of the probability that that test suite is passed, when a particular model of an implementation $i \in \mathit{MODS}$ is given. However, in testing we do not have one particular, given model, but we need to reason about classes of possible models. Some of these possible models are more likely to occur than others, and some are more important than others.

In this section we define a valuation measure on models of implementations. This valuation assigns a value to a set of models of implementations, i.e., to subsets of MODS. This value gives an indication about the importance of the subset as a possible class of correct implementations with respect to a particular specification s and implementation relation **imp**. The valuation is defined analogous to [Bri93] as a measure-theoretic integral, which takes into account both the likeliness of occurrence of the implementations and the importance of the individual implementations, expressed by a probability and a weight on models of implementations, respectively.

Weight of implementations An implementation relation distinguishes between correct and incorrect models of implementations. However, to express the importance of each implementation, more discriminating power has to be added. This is done by assigning a weight to each implementation.

Let $s \in \mathit{SPECS}$ be a specification, and $\mathbf{imp} \subseteq \mathit{MODS} \times \mathit{SPECS}$ an implementation relation, then a function $w : \mathit{MODS} \to \mathbb{R}\backslash\{0\}$ is a weight assignment function on MODS with respect to s and **imp**, if for all $i \in \mathit{MODS}$:

$$w(i) > 0 \iff i \; \mathbf{imp} \; s \tag{11}$$

A weight assignment assigns a positive real number to each conforming implementation, and a negative number to each erroneous implementation. For conforming implementations the weight can express that one implementation is better than another; negative weights express the gravity of errors in erroneous implementations: if $w(i_1) < w(i_2) < 0$ then both i_1 and i_2 are not correct, but the errors of i_1 are more severe than those of i_2. Note that the weight is defined with respect to a specification s and an implementation relation **imp**: it assigns a weight to each implementation as a candidate for an implementation of the particular s and **imp**.

Probability on implementations Given a specification $s \in S$ and an implementation relation **imp** implementers will start developing a concrete implementation $IUT \in IMPS$. Many different implementations, modelled by different models, may occur as the result of the implementation process, some of them conforming, and others nonconforming. Not every possible implementation will have the same chance to occur as the result of implementing a given specification, e.g., when designing a coffee machine which is supposed to serve coffee and tea it is less likely to end up with a completely different kind of machine (e.g., a gambling machine) than to end up with a slightly different, but possibly incorrect, coffee machine. Moreover, assuming that an implementer can make several independent mistakes with non-zero probability it is less likely for the implementer to make all possible mistakes than to make only a small number of mistakes.

Similar to [Bri93] we view the design and implementation process of a complex system as a stochastic experiment that draws a concrete implementation IUT modelled by the stochastic function $\underline{i} \in MODS$ from the sample set of all possible implementations $MODS$ according to a certain probability distribution. Let $MODS$ be discrete, and let the probability density function be given by p_s, i.e., $p_s(i)$ denotes the probability that implementation i occurs as the result of the design and implementation process, then

$$P_s(I) =_{\text{def}} \sum_{i \in I} p_s(i) \tag{12}$$

denotes the probability that the activity of implementing specification s produces an implementation that is modelled by a member of I.

Valuation Using the probability density p_s and the weight assignment w it is possible to define a valuation on discrete sets of implementations, which captures the importance of a set of implementations in terms of their weight and their probability of occurrence:

$$\mu(I) =_{\text{def}} \sum_{i \in I} w(i)\ p_s(i) \tag{13}$$

The expression $\mu(I)$ denotes the value of implementations in $I \subseteq MODS$. It expresses, in some sense, the aggregate correctness of the candidate implementations in I with respect to the given s and **imp**. Note that correct implementations may compensate incorrect implementations (and vice versa). Consequently, $\sum_{i \in I} w(i)\ p_s(i) < 0$ does not imply that the probability of obtaining an erroneous implementation in I is high: the negative value might occur because of one unlikely, but very negatively weighted erroneous implementation. The expectation value of the weight of an arbitrarily designed implementation in $MODS$ is expressed by $\sum_{i \in MODS} w(i)\ p_s(i)$.

Although in most practical cases it suffices to consider the valuation as a, possibly infinite, summation, it is more abstract and general to rewrite (12) and (13) as measure-theoretic integrals, which also encompass continuous domains of implementations. Consider a probability measure P_s on sets of implementations $I \subseteq MODS$ defined by

$$P_s(I) =_{\text{def}} Pr\{\underline{i} \in I\} \tag{14}$$

then, under sufficient assumptions of neatness of the underlying sets (they should be Borel), the valuation μ can be expressed as the measure-theoretic integral

$$\mu(I) =_{\text{def}} \int_I w(i)\ dP_s \tag{15}$$

Proposition 2

1. $\int_\emptyset w(i)\ dP_s = 0$
2. If $I_s \neq \emptyset$ then $\int_{I_s} w(i)\ dP_s > 0$
3. If $\overline{I_s} \neq \emptyset$ then $\int_{\overline{I_s}} w(i)\ dP_s < 0$

□

5 COMPARING TEST SUITES

The purpose of conformance testing is to increase the confidence in the correct functioning of an implementation by detecting whether it conforms to its specification or not. Test-suite execution is expected to reject, i.e., yield the verdict **fail** with, nonconforming implementations, and to accept, i.e., yield the verdict **pass** with, conforming ones. Since a perfect test suite exactly doing this is not likely to be encountered in practice (cf. section 2), a method for comparing the quality of test suites is needed in order to select the best test suite for a given conformance testing problem. A comparison of test suites can be made if a quantitative measure can be assigned to each test suite. Such a measure should quantify the ability of a test suite to reject nonconforming implementations and to accept conforming ones, and it should take into account the probability of occurrence of implementations (a test suite that detects errors which are very likely to occur has higher quality), the weight of implementations (a test suite that detects a nonconforming implementation with severe errors, i.e., with very negative weight, has higher quality), and the probability that an erroneous implementation indeed yields the verdict **fail** (a test suite that rejects an erroneous implementation with higher probability has higher quality).

Soundness and exhaustiveness The probability of occurrence of implementations was expressed by the probability measure P_s on sets of implementations (14), and the probability that an implementation i yields the verdict **pass** with test suite T was expressed by the probability measure $P_v^{T,i}$ (9,10). Under the assumption that the probability of occurrence of implementations (the measure P_s) is independent of the probability of yielding **pass** (the measure $P_v^{T,i}$) we can integrate these measures to obtain the probability measure on $MODS \times \{\textbf{pass}, \textbf{fail}\}$:

$$\pi_T(I, V) =_{\text{def}} \int_I \int_V dP_v^{T,i}\ dP_s \tag{16}$$

$\pi_T(I, V)$ expresses the probability that an implementation $i \in I \subseteq MODS$ occurs, and that i, when tested with T, yields a verdict in $V \subseteq \{\textbf{pass}, \textbf{fail}\}$. So, $\pi_T(I, \{\textbf{pass}\})$ denotes the probability that an implementation $i \in I$ occurs which passes test suite T. Note that the integrals cannot be interchanged, since the inner integral depends on $i \in I$.

Taking also the weight of implementations (section 4) into account we obtain analogously the valuation measure λ:

$$\lambda_T(I, V) =_{\text{def}} \int_I \int_V w(i)\ dP_v^{T,i}\ dP_s \tag{17}$$

Given a test suite T, the valuation λ assigns a value to each pair of $I \subseteq \mathit{MODS}$ and $V \subseteq \{\mathbf{pass}, \mathbf{fail}\}$. Of course, the interesting values of λ are the combinations of conforming and nonconforming implementations with the verdicts **pass** and **fail**:

	$V = \{\mathbf{pass}\}$	$V = \{\mathbf{fail}\}$
$I = I_s$	$\lambda_T(I_s, \{\mathbf{pass}\})$	$\lambda_T(I_s, \{\mathbf{fail}\})$
$I = \overline{I_s}$	$\lambda_T(\overline{I_s}, \{\mathbf{pass}\})$	$\lambda_T(\overline{I_s}, \{\mathbf{fail}\})$

For a good test suite, i.e., one with a large ability to reject nonconforming implementations and to accept conforming ones, the values $\lambda_T(I_s, \{\mathbf{pass}\})$ and $\lambda_T(\overline{I_s}, \{\mathbf{fail}\})$ are optimized, or, equivalently, the values $\lambda_T(I_s, \{\mathbf{fail}\})$ and $\lambda_T(\overline{I_s}, \{\mathbf{pass}\})$ are minimized.

We define two measures on test suites, soundness *snd*, quantifying the ability to accept conforming implementations, and exhaustiveness *exh*, quantifying the ability to reject nonconforming implementations, by normalization of the valuation λ with respect to all conforming implementations, and all nonconforming implementations, respectively. The soundness of test suite T with respect to $I_s \subseteq \mathit{MODS}$ and weight assignment w is

$$\mathit{snd}(T) =_{\text{def}} \frac{\lambda_T(I_s, \{\mathbf{pass}\})}{\lambda_T(I_s, \{\mathbf{pass}, \mathbf{fail}\})} \tag{18}$$

Similarly, the exhaustiveness of test suite T with respect to I_s and w is

$$\mathit{exh}(T) =_{\text{def}} \frac{\lambda_T(\overline{I_s}, \{\mathbf{fail}\})}{\lambda_T(\overline{I_s}, \{\mathbf{pass}, \mathbf{fail}\})} \tag{19}$$

Of course, these definitions are only valid if $\lambda_T(I_s, \{\mathbf{pass}, \mathbf{fail}\}) \neq 0$ and $\lambda_T(\overline{I_s}, \{\mathbf{pass}, \mathbf{fail}\}) \neq 0$, but these requirements are easily satisfied, since they correspond to the existence of conforming and nonconforming implementations, respectively. If they would not exist, there were no need for testing at all.

Proposition 3

1. $0 \leq \mathit{snd}(T) \leq 1$
2. $0 \leq \mathit{exh}(T) \leq 1$
3. T is sound following (4) if and only if $\mathit{snd}(T) = 1$
4. T is exhaustive following (5) if and only if $\mathit{exh}(T) = 1$
5. T is complete if and only if $\mathit{snd}(T) = 1$ and $\mathit{exh}(T) = 1$

□

As noticed in section 4 it mostly suffices to consider summations instead of integrals. Let $p_v^{T,i}$ be the density function corresponding to $P_v^{T,i}$, i.e., $p_v^{T,i}(v) = P_v^{T,i}(\{v\})$, then we can use

$$\lambda_T(I, V) \;=\; \sum_{i \in I} \sum_{v \in V} w(i)\; p_v^{T,i}(v)\; p_s(i) \tag{20}$$

Comparison In the approach of section 2 [ISO96], test suites are compared using the coverage function, which is defined to quantify the extent to which sound test suites are exhaustive. Moreover, test suites are trivially compared with their soundness: test suites which are not sound are simply not considered, i.e., they are worse than any sound test suite. So, in fact, test suites are ordered lexicographically with the pair $\langle \mathit{soundness}, \mathit{coverage} \rangle$, where *soundness* can only take two possible values, viz. sound or not sound.

The two measures *snd* (18) and *exh* (19) generalize the original definitions in (4) and (5): soundness and exhaustiveness are not logical properties anymore, but they are continuous measures on test suites with values between 0 and 1. Similar as above we can now consider an ordering on pairs of the form

$$\langle \mathit{snd}(T), \mathit{exh}(T) \rangle \tag{21}$$

for comparing test suites. The rôle of *exh(T)* is analogous to the coverage function above: it defines a measure of the extent to which a test suite is exhaustive. And indeed, it can be shown that *exh* is a coverage satisfying (6), if *exec* faithfully models concrete test execution (cf. section 2), i.e., if $P_v^{T,i}(\{\mathbf{pass}\}) = 1 \Leftrightarrow i \in P_T$ and $P_v^{T,i}(\{\mathbf{pass}\}) = 0 \Leftrightarrow i \notin P_T$. If this assumption does not hold then *exh* is not a coverage, and it is difficult to define one which satisfies (6).

There are various ways to compare test suites by ordering pairs of the form $\langle \mathit{snd}(T), \mathit{exh}(T) \rangle$: the lexicographical ordering, projections on one of the constituents, addition, vector addition or multiplication of both constituents, comparing the maxima or minima, etc. The actual way of ordering the tuples (21) will depend on the application. In testing the software of a nuclear power plant the exhaustiveness will be the most important: not finding an error if there is one has much more disastrous consequences than finding an error where there is none. If, on the other hand, testing is expensive, then detecting errors where there are none is costly, and should be avoided: soundness will prevail.

6 PROBABILISTIC TESTING WITH INPUTS AND OUTPUTS

In this section we instantiate the framework discussed in the previous sections and illustrate its applicability by a running example. The structure of this section follows the structure of the framework. First, we present some elementary notation for the description of system behaviour as labelled transition systems. Next, we instantiate the universes *SPECS*, *MODS* and *TESTS* as (restricted sets of) labelled transition systems. Then the concepts of conformance and testing (section 2) are instantiated in such a way that they are applicable for the systems under consideration, followed by the concepts defined in section 4 (valuation on implementations). Finally, we discuss how section 5 (comparison) can be instantiated.

Preliminary definitions The behaviour of systems is modelled by means of labelled transition systems. We use the standard definitions of labelled transition systems as can be found in, e.g., [Tre95]: a transition system p is a quadruple $p = \langle S, L, \rightarrow, s_0 \rangle$, where S is a (countable) set of states, L is a (countable) set of observable actions, $\rightarrow \subseteq S \times (L \cup \{\tau\}) \times S$ is a set of transitions, and $s_0 \in S$ is the initial state. The special action $\tau \notin L$ denotes the unobservable action. The universe of labelled transition systems over L is denoted by $\mathcal{LTS}(L)$. A trace σ is a sequence of observable actions ($\sigma \in L^*$), and $\Longrightarrow$ denotes the transition relation between states s, s' when performing trace σ, i.e., $s \overset{\sigma}{\Longrightarrow} s'$ indicates that s' can be reached by performing the observable sequence of actions $\sigma \in L^*$. Furthermore, we define the set of traces by $\mathit{traces}(s) =_{\mathrm{def}} \{\sigma \in L^* \mid \exists s' : s \overset{\sigma}{\Longrightarrow} s'\}$, the set of reachable states from s by $\mathit{der}(s) =_{\mathrm{def}} \{s' \mid \exists s' : s \overset{\sigma}{\Longrightarrow} s'\}$, the set of initial observable actions from $S' \subseteq S$ by $init(S') =_{\mathrm{def}} \{x \in L \mid \exists s' \in S', s \in S : s' \overset{x}{\Longrightarrow} s\}$, the set of reachable states after $\sigma \in L^*$ by s **after** $\sigma =_{\mathrm{def}} \{s' \mid s \overset{\sigma}{\Longrightarrow} s'\}$, and a boolean predicate on states, s **after** σ **refuses** $A =_{\mathrm{def}} \exists s' : s \overset{\sigma}{\Longrightarrow} s'$ and $\forall x \in A : s' \overset{x}{\not\Longrightarrow}$, where $A \subseteq L$. As usual, we will not distinguish between a labelled transition system and its initial state.

Specifications We instantiate the formalism *SPECS* with the set of all labelled transition systems over L, that is, we take $SPECS = \mathcal{LTS}(L)$. Figure 1 depicts a specification $s \in \mathcal{LTS}(L)$ for $L = \{sh, cb, tb, cof, tea\}$. Here, it is assumed that *sh* models the insertion of a shilling, *cb* and *tb* model the pressing of a coffee button and a tea button, respectively, and *cof* and *tea* model the provision of coffee and tea, respectively.

Models of implementations As many systems communicate by means of clearly distinguishable input actions and output actions [Pha94, Tre95], we assume that the set of actions L that a system can perform can be partitioned in a set of input actions L_I and a set of output actions L_U. Furthermore, (viz. [Pha94, Tre95]) we assume that implementations, unlike specifications, never can refuse any input in each reachable state, i.e.,

$$\forall p' \in der(p), \forall a \in L_I : p' \overset{a}{\Longrightarrow} \tag{22}$$

Such systems are called input-output transition systems, and the universe of such systems is denoted by $\mathcal{IOTS}(L_I, L_U)$. We take $MODS = \mathcal{IOTS}(L_I, L_U)$. Note that $\mathcal{IOTS}(L_I, L_U) \subseteq \mathcal{LTS}(L_I \cup L_U)$, thus all input-output transition systems are also labelled transition systems. Figure 1 shows implementations $i_1, \ldots, i_4$ for $L_I = \{sh, cb, tb\}$ and $L_U = \{cof, tea\}$.

Tests A test t over L_U and L_I is a 6-tuple $\langle S, L_U, L_I, \rightarrow, \nu_t, s_0\rangle$, where $\langle S, L_I \cup L_U, \rightarrow, s_0\rangle \in \mathcal{LTS}(L_I \cup L_U)$ is a deterministic, finite-behaviour labelled transition system over L_U and L_I, and $\nu_t : der(s_0) \rightarrow \{\mathbf{pass}, \mathbf{fail}\}$ is a function assigning verdicts to the reachable states of t. When a test is run against an implementation, it is either able to receive all outputs from the implementation, or provide a single input to the implementation, or deadlock. That is, any test satisfies

$$\forall s' \in der(s_0) : init(s') = \{a\} \subseteq L_I \text{ or } init(s') = L_U \text{ or } init(s') = \emptyset \tag{23}$$

The universe of tests over L_I and L_U is denoted by $\mathcal{T}(L_I, L_U)$. We take $TESTS = \mathcal{T}(L_I, L_U)$. Cf. section 3, a test suite is modelled as a multiset of tests. Figure 1 shows tests t_1, t_2, t_3 for $L_I = \{sh, cb, tb\}$ and $L_U = \{cof, tea\}$.

Implementation relation An implementation relation is a relation between the set of specifications *SPECS* and the set of models of implementations *MODS* (section 2). In [Tre95] a family of implementation relations $\mathbf{ioconf}_{\mathcal{F}}$ (for $\mathcal{F} \subseteq (L_I \cup L_U)^*$) is defined that are applicable to the kind of systems we are considering. Informally, an implementation i is a correct implementation with respect to specification s and implementation relation $\mathbf{ioconf}_{\mathcal{F}}$ if for every trace $\sigma \in \mathcal{F}$ each output, or absence of outputs, that the implementation i can perform after having performed sequence σ is specified by s. Let $i \in \mathcal{IOTS}(L_I, L_U), s \in \mathcal{LTS}(L_I \cup L_U)$ and $\mathcal{F} \subseteq (L_I \cup L_U)^*$, then

$$i \ \mathbf{ioconf}_{\mathcal{F}}\ s =_{\mathrm{def}} \forall \sigma \in \mathcal{F} : out(i \ \mathbf{after}\ \sigma) \subseteq out(s \ \mathbf{after}\ \sigma) \tag{24}$$

where $out(p\ \mathbf{after}\ \sigma) =_{\mathrm{def}} (init(p\ \mathbf{after}\ \sigma) \cap L_U) \cup \{\delta \mid p\ \mathbf{after}\ \sigma\ \mathbf{refuses}\ L_U\}$ for $\sigma \in traces(p)$, and $out(p\ \mathbf{after}\ \sigma) =_{\mathrm{def}} \emptyset$ otherwise. Here, the special action $\delta \notin L_I \cup L_U$ models absence of

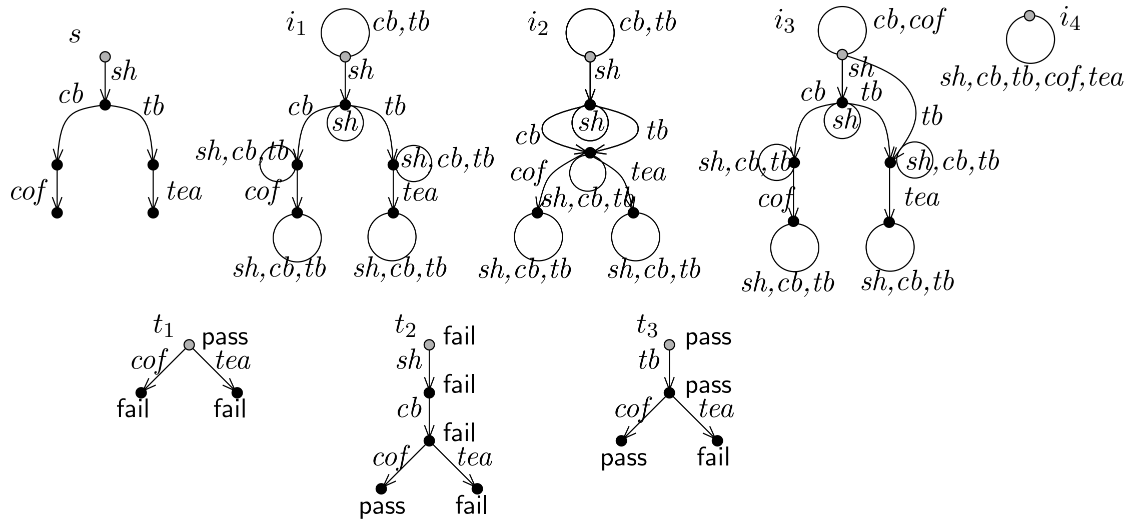

Figure 1: Examples of specifications, implementations and tests.

output actions. If $out(i \textbf{ after } \sigma) \subseteq out(s \textbf{ after } \sigma)$ then we write $i \models_s \sigma$ (and $i \not\models_s \sigma$ for its negation, in which case we say that i has fault σ with respect to s).

Example 4 For specification s (figure 1) we instantiate the implementation relation **imp** (section 2) with $\textbf{ioconf}_{\{\epsilon, sh.cb\}}$. For implementation i_1 we have $out(i_1 \textbf{ after } \epsilon) = \{\delta\} \subseteq \{\delta\} = out(s \textbf{ after } \epsilon)$, and $out(i_1 \textbf{ after } sh.cb) = \{cof\} \subseteq \{cof\} = out(s \textbf{ after } \epsilon)$. Hence $i_1 \textbf{ ioconf}_{\{\epsilon, sh.cb\}}\ s$. However, $out(i_2 \textbf{ after } sh.cb) = \{cof, tea\} \not\subseteq \{cof\} = out(s \textbf{ after } sh.cb)$, i.e. *not* $(i_2 \textbf{ ioconf}_{\{\epsilon, sh.cb\}}\ s)$. Here, i_1 is a correct implementation ($i_1 \in I_s$), and i_2, i_3, i_4 are incorrect ($i_2, i_3, i_4 \in \overline{I_s}$). □

Test execution The running of test $t \in \mathcal{T}(L_I, L_U)$ against implementation $i \in \mathcal{IOTS}(L_I, L_U)$ is modelled by the LOTOS synchronization operator $||$. The execution of a test against an implementation returns a set of observations that may occur, defined by

$$exec(t, i) =_{\text{def}} \{\sigma \in (L_I \cup L_U)^* \mid (t||i) \textbf{ after } \sigma \textbf{ refuses } (L_I \cup L_U)\} \tag{25}$$

(viz. the deadlock traces of $(i||t)$). The set $exec(t, i)$ denotes the set of possible test run outcomes.

Verdict assignment To each set of observations $O \subseteq exec(t, i_{IUT})$ that can be made when a test $t \in \mathcal{T}(L_I, L_U)$ is executed against implementation IUT a verdict has to be assigned (cf. (1)). We instantiate the verdict assignment function by

$$verd_t(O) =_{\text{def}} \begin{cases} \textbf{pass} & \text{if } \forall \sigma \in O : v_t(\sigma) = \textbf{pass} \\ \textbf{fail} & \text{if } \exists \sigma \in O : v_t(\sigma) = \textbf{fail} \end{cases} \tag{26}$$

where v_t denotes the verdict assignment on outcomes $\sigma \in exec(t, i)$ given by

$$v_t(\sigma) = \begin{cases} \textbf{pass} & \text{if } \nu_t(t \textbf{ after } \sigma) = \textbf{pass} \\ \textbf{fail} & \text{if } \nu_t(t \textbf{ after } \sigma) = \textbf{fail} \end{cases} \tag{27}$$

i.e., the verdict **pass** is assigned to a set of outcomes O if and only if each individual test outcome leads to a **pass**-state in test t.

Example 5 Consider test t_2 and implementation i_2, then there are two possible test outcomes: $exec(t_2, i_2) = \{sh.cb.cof, sh.cb.tea\}$. Each outcome leads to a different verdict (viz. (27)) $v_{t_2}(sh.cb.cof) = \nu_{t_2}(t_2 \textbf{ after } sh.cb.cof) = \textbf{pass}$ and $v_{t_2}(sh.cb.tea) = \nu_{t_2}(t_2 \textbf{ after } sh.cb.tea) = \textbf{fail}$. Consequently, if test execution gives us an observation $O = \{sh.cb.cof\}$ then (according to (26)) $verd_{t_2}(O) = \textbf{pass}$, whereas if test execution gives us observation $O = \{sh.cb.cof, sh.cb.tea\}$ then (according to (26)) $verd_{t_2}(O) = \textbf{fail}$. □

Probabilistic test runs If test execution is considered as probabilistic test runs (section 3), a probability distribution $P_o^{t,i}$ is assumed. Consequently, according to (8) a probability distribution $P_v^{t,i}$ on verdicts follows. For simplicity we assume that the probability distribution $P_v^{t,i}$ is given for $t \in \mathcal{T}(L_I, L_U)$ and $i \in \mathcal{IOTS}(L_I, L_U)$. In the next table the set $exec(t, i)$ for $t \in \{t_1, \ldots, t_3\}$ (figure 1) and $i \in \{i_1, \ldots, i_4\}$ (figure 1), together with the set of verdicts $\{v_t(\sigma) \mid \sigma \in exec(t, i)\}$ is given. For each set of verdicts consisting of more than one element, the probability $P_v^{t,i}(\{v\})$ on the occurrence of v is assumed to be known, and denoted behind the verdict. If the set of verdicts contains one element v, then evidently $P_v^{t,i}(\{v\}) = 1$.

		t_1	t_2	t_3
i_1	*exec*	$\{\epsilon\}$	$\{sh.cb.cof\}$	$\{tb\}$
	verd	$\{\textbf{pass}\}$	$\{\textbf{pass}\}$	$\{\textbf{pass}\}$
i_2	*exec*	$\{\epsilon\}$	$\{sh.cb.cof, sh.cb.tea\}$	$\{tb\}$
	verd	$\{\textbf{pass}\}$	$\{\textbf{pass}(1/4), \textbf{fail}(3/4)\}$	$\{\textbf{pass}\}$
i_3	*exec*	$\{cof\}$	$\{sh.cb.cof\}$	$\{tb.tea\}$
	verd	$\{\textbf{fail}\}$	$\{\textbf{pass}\}$	$\{\textbf{fail}\}$
i_4	*exec*	$\{cof, tea\}$	$\{sh.cb.cof, sh.cb.tea\}$	$\{tb.cof, tb.tea\}$
	verd	$\{\textbf{fail}\}$	$\{\textbf{pass}(1/4), \textbf{fail}(3/4)\}$	$\{\textbf{pass}(2/5), \textbf{fail}(3/5)\}$

Example 6 The probability that implementation i_4 passes test case t_2 is 1/4, ($P_v^{t_2,i_4}(\{\textbf{pass}\}) = 1/4$), and the probability that i_4 passes test case t_3 is 2/5 ($P_v^{t_3,i_4}(\{\textbf{pass}\}) = 2/5$). Under the assumption that the outcome of tests does not depend on the outcome of previous (or future) tests, the probability that i_4 passes test suite $\langle t_2, t_3\rangle$ is given by (10): $P_v^{\langle t_2,t_3\rangle,i_4}(\{\textbf{pass}\}) = 1/4 \times 2/5 = 1/10$, and consequently $P_v^{\langle t_2,t_3\rangle,i_4}(\{\textbf{fail}\}) = 9/10$. □

Weight of implementations A weight assignment (11) quantifies the (in)correctness of implementations and can be obtained from weighing the faults that an implementation possesses. The function $w : \mathcal{IOTS}(L_I, L_U) \to I\!R \backslash \{0\}$ defined by

$$w(i) = \begin{cases} 1 & \text{if } i \textbf{ ioconf}_{\mathcal{F}}\ s \\ \sum_{\sigma \in \mathcal{F}} \{\!| g(\sigma) \mid i \not\models_s \sigma |\!\} & \text{otherwise} \end{cases} \tag{28}$$

where $g : \mathcal{F} \to I\!R_{<0}$, and $\{\!| \cdot |\!\}$ denotes a multiset, is a weight assignment function (according to (11)) on $\mathcal{IOTS}(L_I, L_U)$. The function g expresses the gravity of violating $i \models_s \sigma$. Note that the function w is not able to discriminate between correct implementations (all correct implementations are assigned the same value) and that implementations possessing the same faults are assigned the same (negative) weights.

Probability of implementations Let p_σ be the probability that an arbitrary implementation i has fault $\sigma \in \mathcal{F}$, i.e. $i \not\models_s \sigma$, then (under the assumption that all faults are independent)

$$p_s(i) = \prod_{\substack{\sigma \in \mathcal{F} \\ i \models_s \sigma}} (1 - p_\sigma) \times \prod_{\substack{\sigma \in \mathcal{F} \\ i \not\models_s \sigma}} p_\sigma \tag{29}$$

is a probability density function on $\mathcal{IOTS}(L_I, L_U)$. As implementations (dis)satisfying the same requirements are assigned equal probability, p_s can be considered as a probability density function on classes of implementations having the same faults. We shall assume that $i_1, \ldots, i_4$ are representative members of their classes, i.e., it suffices to consider $MODS = \{i_1, \ldots, i_4\}$.

Valuation on implementations Following (13) we can compute the valuation for the set of (in)correct implementations by taking the weights of implementations using (28), and the distribution of the implementations using (29).

Example 7 Let $g(\epsilon) = -5$ and $g(sh.cb) = -3$ then it follows: $w(i_1) = 1, w(i_2) = -3, w(i_3) = -5$ and $w(i_4) = -3 + -5 = -8$. Implementation i_4 violates both requirements imposed by $\mathbf{ioconf}_{\{\epsilon, sh.cb\}}$ and is therefore considered worse compared to all other implementations. Furthermore, assume the probability for arbitrary implementation i to violate the requirement $i \models_s \epsilon$ equals 1/3 (i.e. $p_\epsilon = 1/3$), and the probability to violate the requirement $i \models_s sh.cb$ equals 2/5. Then it follows $p_s(i_1) = (1 - 1/3) \times (1 - 2/5) = 2/5$, $p_s(i_2) = (1 - 1/3) \times 2/5 = 4/15$, $p_s(i_3) = 1/3 \times (1 - 2/5) = 1/5$, and $p_s(i_4) = 1/3 \times 2/5 = 2/15$. From (13) $\mu(\{i_1\}) = 1 \times 2/5 = 2/5$ and $\mu(\{i_2, i_3, i_4\}) = -43/15$. Thus, $\mu(\{i_1, i_2, i_3, i_4\}) = \mu(\{i_1\}) + \mu(\{i_2, i_3, i_4\}) = -37/15$. □

Comparison In the following table we denote the values of $snd(T)$ (18) and $exh(T)$ (19) for some specific T

T	$\lambda_T(I_s, \{\mathbf{pass}\})$	$\lambda_T(\overline{I_s}, \{\mathbf{fail}\})$	$snd(T)$	$exh(T)$
$\langle t_1 \rangle$	2/5	−31/15	1	31/43
$\langle t_2 \rangle$	2/5	−21/15	1	21/43
$\langle t_3 \rangle$	2/5	−41/25	1	123/215
$\langle t_1, t_1 \rangle$	2/5	−31/15	1	31/43
$\langle t_1, t_2 \rangle$	2/5	−40/15	1	40/43

Example 8 Using (20) it follows $\lambda_T(I_s, \{\mathbf{pass}, \mathbf{fail}\}) = 2/5$ and $\lambda_T(\overline{I_s}, \{\mathbf{pass}, \mathbf{fail}\}) = -43/15$ for all test suites T. For test suite $T = \langle t_1, t_2 \rangle$ it follows $\lambda_T(I_s, \{\mathbf{pass}\}) = 1 \times 1^2 \times 2/5 = 2/5$, thus (18) $snd(T) = 1$. Furthermore, $\lambda_T(\overline{I_s}, \{\mathbf{fail}\}) = -3 \times (1 - 1 \cdot (1/4)) \times 4/15 + -5 \times (1 - 0 \cdot 1) \times 1/5 + -8 \times (1 - 0 \cdot (1/4)) \times 2/15 = -40/15$. From (19) it follows $exh(T) = 40/43$. Moreover, $snd(\langle t_2^n, t_3 \rangle) = 1$, and $exh(\langle t_2^n, t_3 \rangle)$ is computed by

$$\frac{-3 \times (1 - (1/4)^n \cdot 1) \times 4/15 - 5 \times (1 - 1^n \cdot 0) \times 1/5 - 8 \times (1 - (1/4)^n \cdot 2/5) \times 2/15}{-3 \times 4/15 + -5 \times 1/5 + -8 \times 2/15}$$

For $n \to \infty$ this yields $\lim_{n \to \infty} exh(\langle t_2^n, t_3 \rangle) = 1$, hence, following proposition (3.5), test suite $\langle t_2^n, t_3 \rangle$ approaches a complete test suite if t_2 is executed infinitely many times for $\mathbf{ioconf}_{\{\epsilon, sh.cb\}}$ with respect to specification s.

Another question might be how often t_2 has to be executed in the test suite $\langle t_2^n, t_3 \rangle$ to reach larger exhaustiveness than $\langle t_1, t_2 \rangle$. This can be calculated by solving the equation $exh(\langle t_2^n, t_3 \rangle) > exh(\langle t_1, t_2 \rangle) = 40/43$: this gives $n > 1$, thus, test suite $\langle t_2, t_2, t_3 \rangle$ has larger exhaustiveness in rejecting erroneous implementations than test suite $\langle t_1, t_2 \rangle$. □

7 CONCLUSIONS

In this paper a probabilistic refinement of [ISO96] with respect to the execution of tests was discussed. In particular, the usually made assumption that every possible outcome that can occur when executing tests against an implementation will be observed (as in, e.g., [ISO96, DNH84]), has been relaxed; it was assumed that outcomes can occur with a certain probability distribution on the set of possible outcomes $exec(t, i)$. This induced a probability distribution on the acceptance or rejection of implementations by test suites.

Furthermore, the notion of correctness has been extended by assigning weights to implementations in order to not only distinguish between correct and incorrect implementations, but also discriminate between different (correct or incorrect) ones by means of their weight (e.g., with respect to the severeness of faults that they possess). Together with a probability distribution on the occurrence of implementations that indicates the probability that a particular (set of) implementation(s) can occur as the result of building a concrete implementation, a valuation on implementations has been obtained. Using such valuations it has been shown that one can quantify the expected correctness value of (sets of) implementations.

By having combined both the probability for a test suite to accept or reject implementations, and the valuation on implementations, normalized measures were obtained, that quantify the extent to which a test suite is able to detect incorrect implementations (19), and the extent to which a test suite is able accept correct implementations (18). Such measures were used to compare, and thus select, test suites with respect to their ability to detect most of the frequently occurring implementations with severe errors, and accept most of the frequently occurring correct systems.

Further work In this paper only one of the assumptions underlying [ISO96] is relaxed. However, [ISO96] states many more assumptions, as stated in section 2, such as the test assumption, the assumption that test cases are correctly implemented, and the assumption that the function *exec* correctly models the process of concrete test execution. Similar as has been done in this paper, one can investigate the consequences when these assumption are relaxed, too.

One of the difficult points in the presented approach is how to obtain the necessary distributions and weight functions. Section 6 gave some possible directions for obtaining them; another way is to use fault domains or fault models. As is indicated in [BDD+92, Bri93] the valuation on implementations (section 4) may be obtained by classifying implementations with respect to a (finite) set of faults or fault models that implementations may possess. In this way the set of implementations $MODS$ is partitioned into classes, such that each class contains all implementations possessing a specific combination of faults. By assigning weights to faults (e.g., by the method proposed by [ACV93]), and assigning probabilities to the occurrence of faults, valuations over classes of implementations can be obtained in a similar way as [BTV91, Bri93]. It needs to be studied how such a partitioning over faults can be used to obtain realistic valuations on implementations.

The applicability of statistical techniques in the current probabilistic extension needs to be investigated. Often the exact distribution with which implementations pass or fail a test is not known apriori because some characteristic parameters of the distribution are not known. In order to estimate the (partially) unknown distribution a sample of the phenomenon under study is taken, and this is used to estimate the unknown parameter(s). Also, the integration of the theory of testing statistical hypotheses such as the acceptance or rejection of the null-hypothesis (H_0 : the system under test is incorrect) and the alternative hypothesis (H_1 : the system under test is correct) and the probabilistic interpretation of test runs as discussed in this paper needs to be explored, in order to ensure that a system is sufficiently tested before considered correct.

In this paper a method is described to obtain a coverage measure before actually executing any test at all. That is, the measures soundness (18) and exhaustiveness (19) can be obtained apriori to the actual test execution process (under the assumption that all relevant parameters, such as distributions P_s and $P_v^{i,T}$ are known in advance). However, one can also measure the assessed soundness (or exhaustiveness) that is obtained after really having executed test suite T. Such measures are called the aposteriori measures. The relation between these two measures has to be investigated, in order to decide whether or not to reject an implementation if the assessed measures are not sufficiently close to the apriori values.

A final remark concerns the continuing discussion about the verdict **inconclusive** in formal testing. Purely semantically, **inconclusive** is equivalent to **pass**: there is no reason to reject the implementation. However, intuitively **inconclusive** refers to not reaching a particular test purpose, if such a test purpose is given apriori. The exact position of **inconclusive**, especially in the context of uncertainty and probability of this paper, remains for further study.

References

[ACV93] J. Alilovic-Curgus and S.T. Vuong. A metric based theory of test selection and coverage. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing, and Verification XIII*, pages 289–304. North-Holland, 1993.

[BDD$^+$92] G. von Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo. Fault models in testing. In J. Kroon, R. J. Heijink, and E. Brinksma, editors, *Fourth Int. Workshop on Protocol Test Systems*, number C-3 in IFIP Transactions, pages 17–30. North-Holland, 1992.

[Bri93] E. Brinksma. On the coverage of partial validations. In M. Nivat, C.M.I. Rattray, T. Rus, and G. Scollo, editors, *AMAST'93*, pages 247–254. BCS-FACS Workshops in Computing Series, Springer-Verlag, 1993.

[BTV91] E. Brinksma, J. Tretmans, and L. Verhaard. A framework for test selection. In B. Jonsson, J. Parrow, and B. Pehrson, editors, *Protocol Specification, Testing, and Verification XI*, pages 233–248. North-Holland, 1991. Also: Memorandum INF-91-54, University of Twente, The Netherlands.

[DNH84] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[ISO96] ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8. *Information Retrieval, Transfer and Management for OSI; Framework: Formal Methods in Conformance Testing.* Committee Draft CD 13245-1, ITU-T proposed recommendation Z.500. ISO – ITU-T, Geneve, 1996.

[LS89] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Sixteenth Annual Symposium on the Principles of Programming Languages.* ACM, 1989.

[Pha94] M. Phalippou. *Relations d'Implantation et Hypothèses de Test sur des Automates à Entrées et Sorties.* PhD thesis, L'Université de Bordeaux I, France, 1994.

[Tre95] J. Tretmans. Testing labelled transition systems with inputs and outputs. In A. Cavalli and S. Budkowski, editors, *Participants Proceedings of the Int. Workshop on*

Protocol Test Systems VIII — COST 247 Session, pages 461–476, Evry, France, September 4-6 1995. Institut National des Télécommunications. Extended abstract of Memorandum INF-95-26, University of Twente, Enschede, The Netherlands, 1995.