A General Testability Theory: Classes, properties, complexity, and testing reductions

Ismael Rodríguez, Luis Llana, and Pablo Rabanal

Abstract—In this paper we develop a general framework to reason about testing. The difficulty of testing is assessed in terms of the amount of tests that must be applied to determine whether the system is correct or not. Based on this criterion, five testability classes are presented and related.
We also explore conditions that enable and disable finite testability, and their relation to testing hypotheses is studied. We measure how far incomplete test suites are from being complete, which allows us to compare and select better incomplete test suites. The complexity of finding that measure, as well as the complexity of finding minimum complete test suites, is identified.
Furthermore, we address the reduction of testing problems to each other, that is, we study how the problem of finding test suites to test systems of some kind can be reduced to the problem of finding test suites for another kind of systems. This enables to export testing methods.
In order to illustrate how general notions are applied to specific cases, many typical examples from the formal testing techniques domain are presented.

Index Terms—Formal testing techniques, general testing frameworks

1 Introduction

Testing consists in checking the correctness of a system by interacting with it. Typically, the goal of this interaction is checking whether an implementation fulfills a property or a specification. If the specification is formally defined then procedures for deriving tests, applying tests, and assessing the outputs collected by tests can be formal and systematic [23], [27], [5], [29], [13]. There exist myriads of formal testing methodologies, each one focusing on checking the correctness of a different kind of system (e.g. *labeled transition systems* [33], [34], [5], [14], *temporal systems* [31], [22], [3], [26], *probabilistic systems* [32], [24], *Java programs* [6], [12], etc). Some methods focus on testing a *part* of the behavior considered critical. Other methods aim at constructing *complete* test suites, that is, sets of tests such that, after applying them to the implementation, the results allow us to precisely determine whether the implementation is correct or not with respect to the specification. For example, checking the correctness of a non-deterministic machine could be impossible regardless of how many tests one applies, because a given behavior could remain hidden for any arbitrarily long time. Even if a machine is deterministic, we could need to apply infinite tests if the number of available ways to interact with the implementation is infinite. In some cases where infinite tests are required, it could be the case that we can achieve any arbitrarily high degree of *partial* completeness with some finite test suite, thus enabling a kind of *unboundedly-approachable* completeness, rather than completeness. Alternatively, if the behavior of the system depends on temporal conditions and the time is assumed to be continuous, then we could need to check what happens when a given input is produced at *all* possible times, thus requiring an uncountable infinite set of tests.

Since the feasibility of a testing method depends on our capability to test systems in finite time, investigating the conditions that enable/disable the existence of finite complete test suites in each case is a major issue in testing. In this regard, several works have studied the effect of assuming some hypotheses about the implementation [18], [29], [19]. By assuming hypotheses, the number of systems that could actually *be* the implementation is reduced, so less tests must be applied to seek for undesirable behaviors. Actually, this may yield a reduction in the number of necessary tests for completeness, from infinite to finite, in some specific cases [2], [15].

However, the general conditions that make the *difference* between requiring infinite or finite sets of tests (regardless of the use of hypotheses) are still unknown. In this paper, by *testability* we will understand how difficult is to test systems, measured in terms of the size required by test suites to be complete. We think that the conditions leading to the testability cases commented on before must be analyzed, that is, whether complete test suites are:

(a) finite;
(b) infinite, but any arbitrarily high degree of partial completeness can be finitely achieved;
(c) countable infinite;

• *I. Rodríguez, L. Llana, and P. Rabanal are with the Department of Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain, 28040. E-mail: isrodrig@sip.ucm.es, llana@sip.ucm.es, prabanal@fdi.ucm.es*

This work is a revised and extended version of CONCUR'09 paper [28].
Work supported by projects TIN2009-14312-C02-01, TIN2012-39391-C04-04, and TIN2012-36812-C02-01.

(d) uncountable infinite;
(e) or there does not exist any complete test suite at all.

To the best of our knowledge, testability scenarios (b), (d), and (e) have never been defined or investigated. A few hypotheses enabling finite complete test suites for some particular models, such as finite state machines, have been reported [23], [27], [5], [29], and some approaches to formally define cases (a) and (c) have been proposed [7], [8]. However, as we will comment later in the related work section, they lack the generality to represent many feasible testing scenarios. Moreover, little is known about the conditions that must hold to make a testing problem fall in each of these cases, as well as the borders between them. This contrasts with other mature fields of Computer Science like Computability or Complexity, where a well established theory allows us to relate different known problems to each other, as well as to classify them into a known hierarchy. Unfortunately, the lack of such formal roots makes the field of Formal Testing Techniques a bit *disorganized*, and techniques are not easily inherited from one problem to another – even if they are quite similar after abstracting factors not directly affecting testability.

In this paper we propose a first step towards the construction of a general testability theory. We will present some formal criteria to classify testing problems according to their testability. Providing a complete testability hierarchy is a huge task and it is out of the objectives of this paper. Instead, we will introduce a hierarchy including only the five main testability cases commented before. We will also present some formal properties of the testability scenario (a) (later called *finitely testable* class) such as conditions required for finite testability, alternative characterizations, transformations keeping the testability, the effect of adding testing hypotheses, methods to *reduce* a testing problem into another, the complexity of finding a minimum complete test suite, and the complexity of measuring the completeness degree of *incomplete* test suites. We will apply these properties to study the testability of some examples. A deeper study of the properties of the remaining four testability classes remains for future work.

1.1 Related work

In this section we present some previous proposals and we use such a presentation to summarize, by the way, the main differences of them with our proposal. As we said before, there is little work dealing with testing from an abstract and general point of view. Among the literature on this subject, it is worth mentioning [7], [8], [10], [30], [35], [25].

In [7], Cherniavsky and Smith propose a recursion theoretic approach to testing, and apply concepts from inductive inference by relating program testing and inductive inference. An abstract formalism to represent testing is presented. Authors show that, for recursively enumerable sets of functions, inference is more difficult than testing, and study the relationship between testable sets and recursively enumerable sets. The proposed model assumes that programs receive a natural number as input and produce another natural number as output. An *adequacy* criterion (for us, testing *completeness*) is defined as follows: Outputs collected by a finite test suite must allow the tester to uniquely determine which model is the IUT (*implementation under test*), given a set of models that could be the IUT. On the contrary, our own notion of completeness will impose a weaker requirement: Given a set of models (functions) representing all possible definitions of the IUT, and a subset representing those that are considered as *correct*, a test suite is complete if outputs after applying it necessarily let us precisely determine whether the IUT belongs to the latter set or not. Thus, determining *which one* is the specific model of the IUT is irrelevant after we have determined on which side of the correctness/incorrectness border the IUT is. Hence, our interpretation of the testing case (a), as proposed before in the introduction, is different. Note that considering a *set* of correct definitions of the IUT, rather than a single correct definition, is natural in many cases: specifications can be partially unspecified and, moreover, sometimes acceptable implementations may be mutually exclusive or denote behaviors which are incompatible with other acceptable implementations.[1] Besides, only deterministic and total functions are considered in [7], whereas our models explicitly denote *non-determinism* (functions may return different non-deterministic outputs, and our completeness criterion requires that the correctness can be determined regardless of which outputs are non-deterministically chosen by the IUT) as well as *non-termination* (e.g. the tester can determine that emitting a and next non-terminating could be undistinguishable from emitting a and next terminating; more generally, our model allows testers to explicitly determine which pairs of outputs can/cannot be distinguished). Regarding the distinction of cases (a)-(e) commented before, in [7] only case (a) is defined (with the previous important differences), and properties given in [7] just focus on comparing testing and inference.

Based on [7], Cherniavsky and Statman developed later a game theoretic approach to testing, as well as a theory of *testing in the limit*, in [8]. Let us note that the proposed notion of *testing in the limit* is not related to case (b) commented before (i.e. unboundedly-approachable testing), but it is strongly related to case (c) (i.e. countable infinite testing). Thus, in [8] the word limit refers to the fact that, if we were able to apply all test cases belonging to an infinite countable set, then the

1. Let us consider a drinks machine that sells orange and lemon juices. It has two buttons, A and B. The machine will be considered correct as long as buttons unambiguously allow users to request orange or lemon drinks. Thus, an implementation where button A *always* returns orange and B *always* returns lemon is correct. Moreover, an implementation where A *always* returns lemon and B *always* returns orange would be correct as well. Any other behavior is not permitted. Both behaviors are incompatible, so considering them as two separate acceptable definitions is a natural choice.

completeness would be achieved. On the contrary, our notion of unboundedly-approachable testing (case (b)) will require that any non-full *completeness measure* can be achieved with some finite test suite, which will be a subtle intermediate point between cases (a) and (c) (in fact, stronger than (c) and weaker than (a)).

In [10], Davis and Weyuker outline a theoretical model for the notion of test adequacy for a given program. Contrarily to our proposal, which will be black-box oriented, a white-box testing approach is considered. A test set is said to be *adequate* for a given program if the input-output behavior of the program for the given test set distinguishes the program from the rest of the programs whose input-output behavior is not identical to the first one. This idea is generalized by providing a *nearness* definition as a measure of the distance between programs. Their model is presented as a programming language that receives one input and generates one output, and a grammar definition is given to measure this distance. Authors also discuss a hierarchy of adequacy notions, based on the distance between two programs, and define the minimal adequation of a set for a program.

Another paper dealing with the complexity of testing different classes of programs in white-box testing is [30], where Romanik and Vitter define a measure of testing complexity called VCP-dimension. In this work, authors give upper and lower bounds on the number of test points needed to determine if a program is approximately correct or to distinguish one program from all other programs in the same class. These test points are given as input/output pairs. This measure is applied to different classes of programs: straight line programs, if-then-else statements, for loops, and the combinations of the previous schemes.

In [35], Zhu and He present a methodology for testing high-level Petri nets in order to test concurrent software systems. The syntax and semantics of the model presented are introduced, and next the model is used to study four different types of testing strategies for Petri nets: transition-oriented, state-oriented, flow-oriented, and specification-oriented. Each method is formally defined by an observation scheme and an adequacy criterion. In particular, the test adequacy is defined in terms of the coverage for the considered criterion.

In [25], Meinke presents a mathematical framework for black-box testing. The model defines a functional requirement as a precondition on the input data and a postcondition on the output data of a system. This approach allows testers to treat coverage, test termination and reliability modeling, as well as test case generation. A program is said to fail a test if the test satisfies the precondition, the program terminates on the input, and the resulting output assignment does not satisfy the postcondition. In this way, Meinke defines program testing as the search for counterexamples of program correctness, and introduces a probabilistic model to compute the probability that a specification fails a test after passing n tests. In this way, the probability of correctness of a program is calculated, which can be used to generate good test cases to find out errors.

1.2 Contributions

Next we summarize the main contributions of this work. These notions will be explained in detail in subsequent sections.

- A general framework to define testing scenarios is given. It provides enough generality to enable the representation of many different testing scenarios, as we will illustrate with many typical examples from the formal testing techniques domain. Our efforts will primarily focus on tackling the most challenging part of testing, that is, guaranteeing that an incorrect IUT will necessarily emit outputs that let us distinguish it from any possible correct definition (which provides *exhaustiveness*), though our completeness notion will not only require this, but also that a correct IUT will emit outputs that let us distinguish it from any possible incorrect definition (which provides *soundness*). In comparison to other general frameworks, ours has the following features:

  1) In order to provide generality, no specific structure of systems (states, transitions, code lines, etc) is assumed. Behaviors are represented by functions relating inputs and outputs, in a black-box manner.

  2) Designers can explicitly denote that there are several correct definitions rather than a single one, which is a natural approach to under-specification and mutually exclusive valid definitions. More importantly, the goal of testing is determining whether the IUT belongs to the set of correct definitions, which is a weaker requirement than uniquely identifying the IUT from a set of possible definitions (as typically considered in the literature).

  3) Testers can explicitly specify which pairs of outputs are (un-)distinguishable through observation. It enables a natural representation of observation issues such as non-termination and imprecise observation of magnitudes.

  4) The non-determinism of systems is explicitly denoted. Moreover, the non-determinism explicitly affects the definition of completeness: A test suite is considered complete only if, after applying it to the IUT, we will always be able to decide whether the IUT is correct or not (regardless of non-deterministic choices actually made by the IUT).

- A classification of testing problems into five classes is provided. This classification includes classes which have been introduced before in the literature but had a different definition (cases (a) and (c)), as well as other classes which have never been proposed to the best of our knowledge, including the class where *unboundedly-approachable* testing is possible (case (b)).
- Properties of the first class (case (a)) are extensively studied, including alternative characterizations and techniques to export methods between testing problems.
- The particular case where the set of possible definitions of the IUT is *finite* is analyzed. The complexity of

problems such as finding the minimum complete test suite, and finding the least hypothesis that makes test suites complete in several scenarios, is studied. Most of them (but not all) turn out to be NP-complete problems.

- In order to put the proposed theoretical ideas into practice, we develop a few elaborated case studies where our framework lets us discover new interesting properties about well-known testing settings (e.g. the inclusion of a part of M. Hennessy's testing framework [17] into case (b)) and other useful models (e.g. the classification of testing *time-out machines* with rational times into the same case (b), and the classification of testing several variants of machines handling *increasing continuous magnitudes* in case (a), which is discovered through a series of testing reductions).

1.3 Paper structure

This paper is structured as follows. In the next section we construct some basic concepts to reason about testability, and in Section 3 we present the classes of our testability hierarchy. Next, in Section 4 we present some properties of the first of these classes, the finite testability class. We focus on studying those aspects that enable/disable finite testability, the complexity of some problems, the effect of assuming testing hypotheses, and the notion of testing reduction. Later, Section 5 presents our case studies. For the sake of readability, the lengthiest proofs (in particular, those concerning results about complexity) are presented in a different section, Section 6. In Section 7 we present the conclusions of the paper and we outline some lines of future work.

2 Testability concepts

In this section we introduce some preliminary concepts and define testability classes. A general notion to denote implementations and specifications is presented. In particular, we introduce an abstraction of computation device (program, hardware system, etc) as a machine that takes an *input* from a set of inputs and computes an *output*. Let us recall that testing consists in observing the outputs of an implementation and comparing them with those allowed by the specification. However, there might be different outputs that are not *distinguishable* under observation, for instance in presence of imprecise measurement or non-termination (see the forthcoming Example 1). Therefore, we need to introduce a notion of *distinguishable outputs*.

Definition 1: Let O be a *set of outputs*.

- A *distinguishing relation* for O is an anti-reflexive symmetric binary relation $\not\doteq$ over O. We denote the complementary of $\not\doteq$ by $\doteq$.
- A *set of distinguished outputs* is a set O of outputs with a distinguishing relation $\not\doteq$.
- We extend the relation $\not\doteq$ to sets of outputs as follows. Let $\mathcal{O}', \mathcal{O}'' \subseteq O$. We say that $\mathcal{O}'$ and $\mathcal{O}''$ *are distinguishable*, denoted by $\mathcal{O}' \not\doteq \mathcal{O}''$, iff $o' \not\doteq o''$ for all $o' \in \mathcal{O}'$ and $o'' \in \mathcal{O}''$. Again, the relation $\doteq$ is the negation of $\not\doteq$.

Hereafter, when no distinguishing relation is explicitly given, we will assume the trivial inequality: $o_1 \not\doteq o_2$ iff $o_1 \neq o_2$. □

The next example illustrates the previous definition.

Example 1: Let us consider a testing scenario where all outputs can be distinguished from each other. Then we may consider a trivial distinguishing relation $\not\doteq$ where two outputs $o_1, o_2 \in O$ are distinguishable iff they are different, i.e. $o_1 \not\doteq o_2$ iff $o_1 \neq o_2$.

This strong distinguishing capability may not be feasible in other frameworks. Let us observe programs which may not terminate, and let us assume that the time at which messages are produced is not represented in collected observations. For instance, the output $o_1 = mes_1 \cdot mes_2 \cdot stop$ denotes that the IUT produces a message mes_1, next produces mes_2, and next terminates ($stop$), whereas the output $o_2 = mes_1 \cdot \perp$ denotes that the IUT produces mes_1 and next is stuck forever, denoted by $\perp$. In practice, o_1 is not effectively distinguishable from o_2 via observation because, if mes_1 is produced, then we cannot guarantee that mes_2 will *not* be observed later, no matter how much time passes. Consequently, we may consider the following (effective) distinguishing relation: We have $o' \not\doteq o''$ iff (i) $o' \neq o''$; *and* (ii) neither $o' = w \cdot \perp$ nor $o'' = w \cdot \perp$, where w is the longest common prefix of o' and o''. Note that if (ii) is not met then o_1 and o_2 are not distinguishable in finite time, so $o_1 \doteq o_2$. In fact, given this definition of $\not\doteq$, if we observe $o \in \{o', o''\}$ and we have $o' \not\doteq o''$ then we can decide whether $o = o'$ or $o = o''$ in finite time.[2] The tester could also represent *deadlocks* and *livelocks* with different symbols if required (e.g. $\perp_1$ and $\perp_2$), and define them as distinguishable or not depending on whether e.g. the tester can monitor consumed processing resources or not.

Now let us assume that the time when each message is produced is denoted in outputs indeed. For instance, the output $\langle (1.2, mes_1), (2.7, mes_2), (3.2, stop) \rangle$ denotes that mes_1 is produced at time 1.2, then mes_2 is produced at time 2.7, and next the system stops at time 3.2. In this case, non-termination issues would not affect the distinguishability of outputs. Let us consider $o = \langle (t_1, o_1), \ldots, (t_n, o_n) \rangle \in O$. If time t_i is reached and o_i is not observed then we know for sure that we are not observing o. Thus, in this case we may use the following criterion: We have $o' \not\doteq o''$ iff $o' \neq o''$.

We could also consider a scenario where two outputs cannot be distinguished if outputs have close, but not identical time stamps. Thus, two observations $o = \langle (t_1, o_1), \ldots, (t_n, o_n) \rangle \in O$ and $o' = \langle (t'_1, o'_1), \ldots, (t'_m, o'_m) \rangle \in O$ would be distinguishable, i.e. $o' \not\doteq o$, iff $n \neq m$ or, for some i, $o_i \neq o'_i$ or $|t_i - t'_i| > \epsilon$ for some given ϵ.

Finally, let us note that the non-distinguishing relation $\doteq$ may be non-transitive. Let us suppose that testers

2. Alternatively, a tester could assume a kind of *non-termination observability* for practical reasons. For instance, the tester could assume non-termination if he observes that some timeout is reached. In that alternative case, he could consider $o' \not\doteq o''$ iff $o' \neq o''$.

check an analog scale, but their observations may deviate $\pm 3\ gr$ from the weight actually indicated by the sensor. Then, we could define our distinguishing relation in such a way that $45\ gr \stackrel{\circ}{=} 47\ gr$ (testers cannot assure that $45\ gr$ and $47\ gr$ observations denote different IUT behaviors indeed) and $47\ gr \stackrel{\circ}{=} 49\ gr$, but $45\ gr \stackrel{\circ}{\neq} 49\ gr$. Similarly, the relation $\stackrel{\circ}{=}$ given in the previous paragraph is not transitive either. □

Next we introduce our notion of computation formalism.

Definition 2: Let I be a set of inputs and O be a set of distinguished outputs. A *computation formalism* C for I and O is a set of functions $f : I \to 2^O$ where for all $i \in I$ we have $f(i) \neq \varnothing$. □

Given a function $f \in C$, $f(i)$ represents the set of outputs we can obtain after applying input $i \in I$ to the computation artifact represented by f. Since $f(i)$ is a set, f may represent a non-deterministic behavior. Besides, C, I and O can be infinite sets. Representing the behavior of systems by means of functions avoids to implicitly impose a specific structure to system models (e.g. states, transitions). Still, elaborated behaviors can be represented. The next example shows different computation formalisms.

Example 2: Let C be a computation formalism representing the set of all (possibly non-deterministic) Mealy machines (also known as *finite state machines*, FSMs). Let M be an FSM and I' and O' be the set of inputs and the set of outputs of M, respectively. M is represented in C by a function $f \in C$ such that we have $f(\sigma) = \{\sigma'_1, \ldots, \sigma'_n\}$ if and only if $\{\sigma'_1, \ldots, \sigma'_n\}$ is the set of sequences of O' outputs that can be answered by M when it receives the sequence σ of I' inputs. For instance, if $\sigma = a \cdot b$, $\sigma' = x \cdot y$, and $\sigma'' = w \cdot z$, then $f(\sigma) = \{\sigma', \sigma''\}$ means that f represents an FSM M where, in particular, if $a \cdot b$ is given then the machine can answer either $x \cdot y$ or $w \cdot z$. Hence, the set I considered in Definition 2 (respectively, the set O) is the set of all *sequences* of symbols belonging to I' (resp. to O'), that is, $I = I'^*$ and $O = O'^*$.Thus, even if I' and O' are finite, I and O are infinite sets. Alternatively, if systems were assumed to be *deterministic* FSMs, then all functions representing a non-deterministic FSM would be removed from C. Other restrictions over the set of systems that are represented by C could be considered (e.g. considering only FSMs with less than n states for some given $n \in \mathbb{N}$, etc).

Now, let us consider a computation formalism C that represents interactive programs written in a given programming language. The interaction of a user with the program can be represented in a similar way as in Example 1. For example, the behavior of a given program P could be denoted by a function f such that, in particular:

$$f\left(\begin{array}{c}\langle(0.5, but_1),\\ (1.25, but_2)\rangle\end{array}\right) = \left\{\begin{array}{l}\langle(0.75, mes_1), (1.5, mes_2), (2, stop)\rangle,\\ \langle(0.75, mes_1), (1.5, mes_3), (2, mes_4),\\ \quad (2.5, stop)\rangle,\\ \langle(0.75, mes_1), \bot\rangle\end{array}\right\}$$

meaning that if the user presses button 1 at time 0.5 and button 2 at time 1.25, then she will receive message 1 at time 0.75 for sure, and next the program non-deterministically chooses one of the following choices: (a) answering message 2 at time 1.5 and stopping at time 2; (b) answering message 3 at time 1.5, giving message 4 at time 2, and then finishing at time 2.5; or (c) not terminating (denoted by the $\bot$ symbol; the effect of non-terminating behaviors on testing will be further discussed later). For example, we have $\langle(0.5, but_1), (1.25, but_2)\rangle \in I$ and $\langle(0.75, mes_1), (1.5, mes_2), (2, stop)\rangle \in O$. Alternatively, if temporal issues were not considered relevant for assessing the correctness of behaviors, then time stamps could be removed from the proposed representation, or they could be replaced by *ordering* stamps denoting the relative order of each O' output with respect to I' inputs. If other factors were considered relevant, additional details could be denoted.

If two FSMs (resp., two programs) produce the same sets of outputs for all inputs then both machines are represented by the same function $f \in C$. This is because a function belonging to C represents a *relation* between inputs and outputs (but not the internal structure of the machine leading to this behavior). □

Computation formalisms will be used to represent the set of implementations we are considering in a given testing scenario. Implicitly, a computation formalism C represents a *fault model* (i.e. the definition of what can be wrong in the *implementation under test*, IUT) as well as the *hypotheses* about the IUT the tester is assuming. For instance, if the IUT is assumed to be a deterministic FSM and to differ from a given correct FSM in at most one transition, then only functions denoting the behaviors of such FSMs (including the correct one) are in C. Alternatively, if we only assume that the IUT is represented by a deterministic FSM, then C will represent all deterministic FSMs.

Computation formalisms will also be used to represent the subset of specification-compliant implementations. Let C represent the set of possible implementations and $E \subseteq C$ represent the set of implementations fulfilling the specification. The goal of testing is interacting with the IUT so that, according to the collected responses, we can decide whether the IUT actually belongs to E or not. Typically, we apply some tests (i.e., some inputs $i_1, i_2, \ldots \in I$) to the IUT so that observed results $o_1 \in f(i_1)$, $o_2 \in f(i_2)$, ... allow us to provide a verdict. Since all outputs are returned by the same function f, we are implicitly assuming that all input applications are *independent*, i.e. the result after applying i_1 does not affect the output observed next, when we apply i_2.[3] Note that dependencies between consecutive input applications can still be represented by using sequences,

3. In practice, this is equivalent to assuming the existence of a *reliable reset* button, a typical testing assumption. Since a reset button allows to recover the initial state after each test application, it allows to reason independently about each test application in a sequence of tests applications.

as in Example 2. For instance, let i_1 and i_2 be inputs such that, if we apply input i_1 before i_2, then we obtain an output o_1; however, if we apply *only* input i_2, then we obtain a different output o_2. In this case, the inputs of the computation formalism would not be system inputs but sequences of system inputs, and the system would be represented by a function f such that $f(i_1 \cdot i_2) = \{o_1\}$ and $f(i_2) = \{o_2\}$.

Definition 3: Let C be a computation formalism. A *specification* of C is a set $E \subseteq C$. □

If $f \in E$ then f denotes a *correct* behavior, whereas $f \in C \backslash E$ denotes that f is incorrect. Thus, a specification implicitly denotes a correctness criterion. For example, let $f, f' \in C$ be two functions such that for all i we have $f(i) = \{a\}$ and $f'(i) = \{b\}$. Then, $E = \{f, f'\}$ denotes that only machines producing *always* a or *always* b are considered correct. We can also construct E in such a way that a given underlying semantic relation is considered (e.g. bisimulation, testing preorder, traces inclusion, conformance testing, etc). For instance, given some $f \in C$, let us consider that f is correct and we wish to be consistent with respect to a given semantic relation $\preceq$, where $A \preceq B$ means that A is correct with respect to B. Then we could define E as follows: $E = \{f' | f' \preceq f \ \wedge \ f' \in C\}$.

Next we identify test suites and complete test suites. By applying tests cases in a *test suite* (i.e. a set of inputs $\mathcal{I} \subseteq I$) to an IUT, collected outputs should (ideally) let us determine whether the IUT is correct or not, i.e. whether the function denoting its behavior belongs to a given specification set $E \subseteq C$. Unfortunately, in black-box testing we do not have access to the IUT definition, so the only thing we can do is to observe the outputs given by the IUT when tests are applied, and collect produced outputs. If observed outputs can be produced by some possible correct behavior (i.e. some $f \in E$) but *cannot* be produced by some possible incorrect behavior (i.e. some $f \in C \backslash E$), then we know for sure that the IUT is correct; otherwise, we cannot assure the IUT correctness. If a test suite $\mathcal{I} \subseteq I$ is such that observed outputs will necessarily let us decide the (in-)correctness of the IUT, then we say that the test suite is *complete*. This requires that, for any pair of correct and incorrect functions (i.e. $f \in E$ and $f' \in C \backslash E$, respectively), there must be a test $i \in \mathcal{I}$ that distinguishes f and f'. Let f'' be the actual behavior of the IUT. When tests $i_1, i_2, \ldots \in \mathcal{I}$ are applied to f'', some outputs $o_1, o_2, \ldots \in O$ are (in general non-deterministically) chosen by f'' and observed $(o_1 \in f''(i_1), o_2 \in f''(i_2), \ldots)$. If $\mathcal{I}$ is complete then, for each pair $f \in E$ and $f' \in C \backslash E$ of possible correct and incorrect behaviors of the IUT, some of these outputs, say o_j, must let us distinguish between f and f'. Assuring this property *a priori*, i.e. regardless of non-deterministic choices actually taken by the IUT, requires that, for some i_j, *all* possible non-deterministic output responses of f must be pairwise distinguishable from all possible non-deterministic output responses of f'. In this case, outputs collected by applying $\mathcal{I}$ to the IUT will let us determine the (in-)correctness of the IUT in any case.

Let us note that, as explained before in Section 1.1, our completeness notion differs from several classical *adequacy* notions given in the literature, where non-determinism is not taken into account and, more importantly, a quite stronger condition is required: rather than determining whether the IUT behavior belongs to E or not, it is required that we are able to distinguish any candidate behavior from any other candidate behavior from a given set. Note that, given $f_1, f_2 \in E$ (respectively, $f_1, f_2 \in C \backslash E$), our notion does not require that f_1 and f_2 are distinguishable, so our requirement is weaker. Besides, defining specific distinguishing relations $\not\cong$ enables subtler relations between observable outputs.

Definition 4: Let C be a computation formalism for I and O, $E \subseteq C$ be a specification, and $\mathcal{I} \subseteq I$ be a set of inputs.

We say that $f \in E$ and $f' \in C \backslash E$ are *distinguished* by $\mathcal{I}$, denoted by $\texttt{di}(f, f', \mathcal{I})$, if there exists $i \in \mathcal{I}$ such that $f(i) \not\cong f'(i)$.

We say that $\mathcal{I}$ is a *complete test suite* for C, E if for all $f \in E$ and $f' \in C \backslash E$ we have $\texttt{di}(f, f', \mathcal{I})$. □

Note that, in the previous definition, $\not\cong$ is applied to *sets* of outputs, so all outputs from a set are required to be distinguishable from all outputs from the other set.

Observations collected by complete test suites precisely determine whether the IUT is correct or not. Let $\mathcal{I}$ be a complete test suite for the computational formalism C and the specification E. Then, we trivially conclude that:

(A) if $f \in E$, then there is no $f' \in C \backslash E$ such that $f'(i) \stackrel{\circ}{=} f(i)$ for all $i \in \mathcal{I}$, and

(B) reciprocally, if $f \in C \backslash E$ then there does not exist $f' \in E$ such that $f'(i) \stackrel{\circ}{=} f(i)$ for all $i \in \mathcal{I}$.

That is, after applying $\mathcal{I}$, no correct IUT can be considered as incorrect, and no incorrect IUT can be considered as correct.

3 Testability hierarchy

Once the basic general framework has been presented, we introduce four of the five classes of our testability hierarchy: `Class I`, `Class III`, `Class IV`, and `Class V` (`Class II` will be defined later).

Definition 5: Let C be a computation formalism for I and O, and $E \subseteq C$ be a specification.

We say that a pair (C, E) is *finitely testable* if there exists a finite complete test suite for C, E. We denote the set of all finitely testable pairs (C, E) by `Class I`.

We say that (C, E) is *countably testable* if there exists a countable complete test suite for C, E. We denote the set of all countably testable pairs (C, E) by `Class III`.

We say that (C, E) is *testable* if there exists a complete test suite for C, E. We denote the set of all testable pairs (C, E) by `Class IV`.

We denote the set of all pairs (C, E) by `Class V`. □

This classification induces the relation `Class I` $\subseteq$ `Class III` $\subseteq$ `Class IV` $\subseteq$ `Class V`. `Class II`, whose

definition is harder, will be presented in Section 3.2, Definition 7. Next we show some examples fitting into each of these testability classes. They show that all the inclusions considered in the previous relation are proper indeed. Besides, the proposed example of a pair (C, E) in `Class IV` but not in `Class III` will be used to argue that the interest of `Class IV` is, perhaps, just theoretical.

Example 3: Given $n \in \mathbb{N}$, let C_1 represent the set of all deterministic completely-specified FSMs with at most n states where the finite sets of inputs and outputs are I' and O', respectively. Let $E_1 \subseteq C_1$, and $\mathcal{I}_1$ be the set of all sequences of I' symbols whose length is at most $2n+1$. It is known that, if two FSMs M_1 and M_2 represented by C_1 produce different responses for some input sequence, then sequences answered by M_1 and M_2 are different for at least one sequence belonging to $\mathcal{I}_1$ [9], [23]. Hence, for all $f \in E_1$ and $f' \in C_1 \backslash E_1$ there exists a sequence $i \in \mathcal{I}_1$ allowing to distinguish f and f'. Thus, $(C_1, E_1) \in$ `Class I`. As we will see in Example 6 (used to illustrate notions of Section 4) this result can also be proved by applying Lemma 1 (c), which makes this result straightforward and avoids the necessity of identifying any specific complete test suite $\mathcal{I}_1$.

We consider again the previous case, but this time we remove the restriction that FSMs have at most n states. Let C_2 be the resulting computation formalism, and let $E_2 \subseteq C_2$. It is known that, in general, given two (possible infinite) sets of deterministic FSMs, there is no finite set of sequences $\mathcal{I}_2$ allowing to distinguish each member of the first set from each FSM in the other set. Hence, in general we have $(C_2, E_2) \notin$ `Class I` (in Example 6 we also prove this property, this time by using forthcoming Lemma 1 (b)). However, the set of all sequences of inputs from I', that is I'^*, distinguishes all pairs of non-equivalent deterministic FSMs. In fact, I'^* can be numbered (e.g. lexicographically). Thus, we have $(C_2, E_2) \in$ `Class III`.

Let us consider a variant of deterministic FSMs where the transition taken at each situation depends on the elapsed *time*, and let the time be assumed to be continuous. For each state and input, a machine may have several outgoing transitions, each one labeled by a *time interval* $[t_1, t_2]$ with $t_1, t_2 \in \mathbb{R}$ such that $t_1 \leq t_2$. When an input i is received by the machine, the machine takes the first defined transition reacting to i such that the elapsed time fits into its interval. We denote by C_3 the proposed computation formalism, and we consider $E_3 \subseteq C_3$. The set of all input sequences produced at *all* possible real times is a complete test suite for (C_3, E_3), so $(C_3, E_3) \in$ `Class IV`. However, in general we have $(C_3, E_3) \notin$ `Class III`. In order to see this, let us note that the IUT could have some faulty transitions with intervals of the form $[t, t]$ with $t \in \mathbb{R}$, and detecting such transitions requires considering all input sequences in all *real times*.

Now, let C'_3 be defined as C_3, but only intervals following the form $[t_1, t_2]$, with $t_1 < t_2$ and $t_1, t_2 \in \mathbb{R}$, are allowed in machines represented by C'_3. Let $E'_3 \subseteq C'_3$. In this case, the set of all input sequences produced at all possible *rational* times is complete for C'_3, E'_3: For all $t_1, t_2 \in \mathbb{R}$ with $t_1 < t_2$ there exists a rational t with $t_1 < t < t_2$, so by emitting all input sequences in all possible rational times we will be able to observe all IUT transitions. Since the set of all rational numbers is countable and the set of all finite input sequences is so, the set of all finite sequences of rational-timed inputs is countable as well (note that if A is countable then A^* is countable, and if B is countable too then $A \times B$ is countable as well). Hence, we have $(C'_3, E'_3) \in$ `Class III`.[4]

We could argue that the problematic intervals of C_3, i.e. those of the form $[t, t]$ with $t \in \mathbb{R}$, are artificial because we cannot produce an input at the (exact) time t. As far as we have analyzed other examples in `Class IV` but not in `Class III`, they are so for similar reasons. Thus, the practical utility of `Class IV`, if it has any, has not been proved yet. However, we include `Class IV` in our hierarchy for the sake of theory completeness.

Let C_4 be a computation formalism representing all non-deterministic (non-timed) FSMs, and let $E_4 \subseteq C_4$. We consider an FSM M_1 that answers b when a is received, and another FSM M_2 which behaves in the same way as M_1 for any input different from a, and behaves non-deterministically for a: If M_2 receives a, then M_2 can answer either b or c. Let us suppose that M_1 is *correct* and M_2 is not, and let they be represented in C_4 by $f \in E_4$ and $f' \in C_4 \backslash E_4$, respectively. Despite the fact that we *could* obtain c after applying a to f', input a does not necessarily distinguish f and f' because both of them could produce b in response to a. In fact, M_2 could hide the output c for any arbitrarily long time, no matter how many times we apply a. Thus, no input allows the tester to necessarily distinguish f and f', so we have $(C_4, E_4) \in$ `Class V` but $(C_4, E_4) \notin$ `Class IV`.[5]

Non-determinism does *not* imply that finite testability is not possible. Let us consider $C_5 = \{f, f'\}$ and $E_5 = \{f\}$ be such that $f(a) = \{x\}$, $f'(a) = \{x, y\}$, $f(b) = \{x, y\}$, and $f'(b) = \{z, w\}$. Then, $\{b\}$ is a complete test suite for C_5 and E_5. Thus, $(C_5, E_5) \in$ `Class I`.

Finally let us note that $(C, E) \in$ `Class I` does not imply that C or E are finite, or even that C and E represent some simple computation formalisms such as e.g. FSMs. Let C represent all deterministic terminating functions from integers to integers, written in some programming language, that are either strictly increasing or strictly decreasing. Let $E \subseteq C$ consist of all strictly increasing functions in C. Though C and E are infinite sets and

4. Besides, there are in the literature some specific temporal systems dealing with continuous time where testing completeness can be finitely achieved. This is the case for *Timed Automata* under the assumptions that the number of states is known, and upper and lower bounds of all temporal intervals are required to be *integers* [31]. Then, the time can be discretized into a finite set of time regions, and completeness can be finitely achieved by considering only these regions for testing purposes.

5. If *fairness* were assumed, we would be considering a *different* computation formalism $C'_4 \neq C_4$. In C'_4, for all single input denoting a long repetition of the same experiment, the single output would denote that all possible reactions are observed at least once.

FSMs cannot express all functions in C or all functions in E, $\{3,5\}$ is a complete test suite for (C,E): For all $f \in C$, $f(3) < f(5)$ iff $f \in E$. So, $(C,E) \in$ `Class I`. This trivial example shows that the finite testability does not lie in the finiteness or the simplicity of C or E, but in the simplicity of the *border* between correctness and incorrectness. □

If $(C,E) \in$ `Class I` then we can precisely decide if the IUT is correct or not (i.e., if it belongs to the specification set) by considering the answers collected by a complete test suite. On the contrary, if the problem belongs to `Class III` or `Class IV`, but not to `Class I`, then applying a complete test suite to the IUT is unfeasible because the suite is infinite. In particular, if the problem belongs to `Class III` then we could rather speak about testability *in the limit*, i.e. as we iteratively apply more tests, we could tend to complete testing coverage (later this idea will be further elaborated to define `Class II`, given in Definition 7). This is not the case for problems in `Class IV` but not in `Class III`: Applying an *infinite* number of tests one after each other in a given order is not enough to reach complete coverage, because the complete test suite is not countable.

3.1 Non-termination

Next we discuss the relation between complete test suites and non-termination. Let us consider a computation formalism C where systems may not terminate, and let $\mathcal{I}$ be a finite complete test suite. Each correct function is distinguished from each incorrect function for some $i \in \mathcal{I}$ (and vice versa). Let us suppose that the distinguishing relation $\not\approx$ is defined in such a way that two outputs are distinguished only if we can distinguish them in *finite* time (e.g. as we did in Example 1, second paragraph). Let $f \in C \backslash E$ (the argument if $f \in E$ is symmetric). If we apply all inputs in $\mathcal{I}$ to f then, for some of these inputs, we will observe some outputs allowing to distinguish f from all correct functions. By the definition of $\not\approx$, for all of these inputs the part of the answer needed to make the distinction is reached in finite time. However, for those inputs of $\mathcal{I}$ *not* distinguishing function f from any correct function, f could *not* terminate (this is the case if the specification *permits* not to terminate for some inputs). Thus, the procedure consisting in applying all inputs of $\mathcal{I}$ *one after each other* could not terminate indeed.

Nevertheless, let us note that we could always return verdicts in finite time if all inputs in $\mathcal{I}$ were applied to (several instances of) the IUT in *parallel* or *interleaving*: Since each required distinction is provided in finite time, once all distinctions are reached we can stop the parallel/interleaved execution of tests and provide a verdict. Alternatively, we could adopt a more conservative approach where complete test suites are not allowed to include any input i such that, for some $f \in C$, $f(i)$ could not terminate. In this case, the application of a complete test suite terminates even if inputs are applied sequentially. The results presented later in Section 4 also apply to this alternative notion of complete test suite, once a straightforward adaptation is made in each case.

3.2 Definition of `Class II`

In this section we elaborate on our idea of finite testability *in the limit* or, more precisely, *unboundedly-approachable* finite testability. When finite testability is not possible, in some cases it may still be possible to test the IUT up to any arbitrarily high *distinguishing rate* with a *finite* test suite. As distinguishing rate we will consider the *ratio* between the number of pairs of correct/incorrect functions that are distinguished by the test suite and the total number of correct/incorrect pairs. Let us note that this ratio can only be defined if the computation formalism C is finite.

Definition 6: Let C be a finite computation formalism for I and O, $E \subseteq C$ be a specification, and $\mathcal{I} \subseteq I$ be a set of inputs. We define the *distinguishing rate* of $\mathcal{I}$ for (C,E), denoted by $\texttt{d-rate}(\mathcal{I},C,E)$, as:

$$\frac{|\{(f,f') | f \in E, f' \in C \backslash E, \texttt{di}(f,f',\mathcal{I})\}|}{|E| \cdot |C \backslash E|}$$

□

The assumption that C is finite (otherwise the previous division would be undefined) reduces the applicability of the previous notion, as many interesting computation formalisms are infinite indeed. In order to adapt this idea to infinite computation formalisms, we consider *sequences* of finite computation formalisms. Let us note that, if C is a countable set, then there exists a non-decreasing chain of finite subsets:

$$C^1 \subseteq C^2 \subseteq \cdots \subseteq C^k \subseteq C^{k+1} \subseteq \cdots$$

such that $C = \bigcup_{i \in \mathbb{N}} C^i$. Given a specification $E \subseteq C$, let us consider the sub-specifications $E^i = E \cap C^i$. Since each C^i and E^i are finite sets, we can apply Definition 6 to them indeed. Then, the class of pairs (C,E) which are *unboundedly-approachable by finite testing* (`Class II`) consists of those pairs such that, for some non-decreasing sequence like those described above, any arbitrarily high distinguishing rate less than 1 is reached, by some finite test suite, for *almost all* computation formalisms in the sequence.

Definition 7: We say that (C,E) is *unboundedly-approachable by finite testing* if there is a non-decreasing sequence: $C^1 \subseteq C^2 \subseteq \cdots \subseteq C^k \subseteq C^{k+1} \subseteq \cdots$ with $C = \bigcup_{i \in \mathbb{N}} C^i$ such that, for all $\epsilon < 1$, there exists a finite set of inputs $\mathcal{I} \subseteq I$ and $n \in \mathbb{N}$ such that, for all $l \geq n$, $\texttt{d-rate}\left(\mathcal{I}, C^l, E^l\right) \geq \epsilon$, where $E^l = C^l \cap E$ for all $l \in \mathbb{N}$.

We denote by `Class II` the set of all pairs (C,E) that are unboundedly-approachable by finite testing. □

It is straightforward to see that this class is related with the other classes as expected: `Class I` $\subseteq$ `Class II` $\subseteq$ `Class III`. Moreover, the next two examples show that both inclusions are proper.

Example 4: We revisit (C_2,E_2) as defined before in Example 3. In particular, let us assume that the sets of

FSM inputs and outputs are $I' = \{a, b\}$ and $O' = \{0, 1\}$, respectively, and $E_2 = \{f_1\}$ consists of a single function f_1. Next we prove that $(C_2, E_2) \in$ `Class II`, i.e. (C_2, E_2) is unboundedly-approachable by finite testing.

First we define a non-decreasing sequence of finite sets $C^0 \subseteq C^1 \subseteq C^2 \subseteq \cdots C^n \subseteq C^{n+1} \subseteq \cdots$ as required by the definition of `Class II`. Let us introduce some notation. We say that $f \in C_2$ is the *smallest* function fulfilling a given criterion Q if f fulfills Q and, for all function $f' \in C_2 \backslash \{f\}$ fulfilling Q, the smallest FSM (i.e. the one with less states) behaving as defined by f has less states than the smallest FSM behaving according to f', or both have the same number of states but f is smaller according to some fix arbitrary criterium (e.g. we can sort tied functions lexicographically according to their respective infinite sequences $f(0) \cdot f(1) \cdot f(00) \cdot f(01) \cdot f(10) \cdot f(11) \cdot f(000) \cdot \ldots$). Note that, for any set of functions, there always exists the smallest function fulfilling Q (as long as some function fulfills Q).

We build each set C^n as follows. First, let $C^0 = E_2 = \{f_1\}$. Besides, for all $n \geq 1$, the set C^n is constructed as follows. Let us consider the set of all combinations of output responses that a deterministic FSM could give for all input sequences in $\{a, b\}^n$. For instance, if $n = 2$, then all possible responses to sequences aa, ab, ba, bb consist in answering 00, 00, 00, 00 respectively, 00, 00, 00, 01 respectively, ..., 11, 11, 11, 11 respectively. Note that not all combinations of 8 bits are possible indeed. For instance, the combination 00, 10, 00, 00 is impossible because we cannot have $f(aa) = \{00\}$ and $f(ab) = \{10\}$: It would imply that, at its first state, the FSM can answer either 0 or 1 when a is provided, but the FSM must be deterministic. According to these notions, C^n is constructed as follows: For each combination of responses to $\{a, b\}^n$ which is possible in a deterministic FSM, we introduce in C^n the *smallest* function providing that combination *if* f_1 does not provide that particular combination, else f_1 is introduced instead; and there are no other functions in C^n.

Let us show that, for all $i \geq 0$, we have $C^i \subseteq C^{i+1}$. Let $f \in C^i$. This implies that either $f = f_1$ or f is the smallest function providing some combination of responses to $\{a, b\}^i$. If $f = f_1$ then $f \in C^{i+1}$, because f_1 also belongs to C^{i+1}. Let $f \neq f_1$. Thus f is the smallest function providing some combination of responses to $\{a, b\}^i$. Note that f also provides *some* combination of responses to $\{a, b\}^{i+1}$ and, moreover, it is also the smallest function providing *that* combination. We prove it by contradiction: If the smallest function for that combination of responses to $\{a, b\}^{i+1}$ were $f' \neq f$, then f' would be smaller than f, so f' would also be the smallest function providing the same combination as f for $\{a, b\}^i$. However, the smallest function for that combination in $\{a, b\}^i$ is f indeed, so we have a contradiction. Thus, f also belongs to C^{i+1}.

Let us calculate $|C^n|$ for each $n \geq 1$. We have that C^n has as many elements as possible combinations of responses to $\{a, b\}^n$ exist in a deterministic FSM. Let us consider a binary tree with depth n, where left arcs are labeled by a and right arcs are labeled by b. Sequences of labels in each branch, from the root to a leaf, represent each sequence in $\{a, b\}^n$. Each combination of responses of a deterministic FSM to all of these sequences can be represented by assigning a value in $\{0, 1\}$ to each arc in the tree. Since there are $2^{n+1} - 2$ arcs in the tree, there are $2^{2^{n+1}-2}$ possible assignments. Thus, $|C^n| = 2^{2^{n+1}-2}$.

We also have to prove $C_2 = \bigcup_{n \in \mathbb{N}} C^n$. Let $f \in C_2$. Let us suppose that the smallest deterministic FSM behaving as f has k states. Note that the number of functions which can be represented by deterministic FSMs with k or less states is finite, let K be this number. If $f = f_1$ then $f \in C^0$. Let $f \neq f_1$. By contradiction, let us suppose that $f \notin C^i$ for all $i \in \mathbb{N}$. This implies that there does not exist $i \in \mathbb{N}$ such that, for the combination of output sequences answered by f to $\{a, b\}^i$, we have both that (a) this combination differs from the combination answered by f_1 to $\{a, b\}^i$; and (b) f is the smallest function providing that combination. Since $f \neq f_1$, there exists i' such that, for all $i > i'$, the combination of sequences responded by f to $\{a, b\}^i$ differs from the combination given by f_1 (otherwise we would have $f = f_1$ indeed). Thus, for all $i > i'$ (b) is not met, i.e. f is not the smallest function providing its own combination of responses to $\{a, b\}^i$. For each $i > i'$, let f^i be the smallest function providing the combination of responses given by f to $\{a, b\}^i$. Since we are assuming $f \notin C^i$ for all $i \in \mathbb{N}$, we have $f^i \neq f$ for all $i > i'$. Note that, for all $i > i'$, there must exist $j > i$ such that $f^i \neq f^j$ (otherwise f and f^i would have the same behavior for all $\{a, b\}^j$ with $j \in \mathbb{N}$, so we would have $f = f^i$). Let $j_1 < j_2 < \ldots$ be an infinite sequence of naturals such that $f^{j_h} \neq f^{j_{h+1}}$ for all h. Since there exist at most K functions which are smaller than f, we have that $f^{j_{K+1}}$ cannot be smaller than f. Thus we have a contradiction, and we infer $f \in C^{j_{K+1}}$.

Let us prove $(C_2, E_2) \in$ `Class II`. Hereafter we will assume $s_n = |C^n|$. Let $\varepsilon < 1$. We find a test suite $\mathcal{I}$ and a natural $n \in \mathbb{N}$ such that the distinguishing rate of $\mathcal{I}$ is higher than or equal to ε for all C^l with $l \geq n$. For all $k \in \mathbb{N}$, let $\mathcal{I}^k = \{a, b\}^k$. Besides, let us consider the equivalence relation $f \equiv_k f'$ iff $f(\sigma) = f'(\sigma)$ for all $\sigma \in \mathcal{I}^k$. For any $l > k$, the relation $\equiv_k$ is a equivalence relation in C^l. The class of equivalence of a function f (that is, the set of functions f' such that $f \equiv_k f'$) will be denoted by $[f]$. Note that there are exactly s_k equivalence classes in C^l, and each class has $\frac{s_l}{s_k}$ elements. Moreover, since $E_2 = \{f_1\}$, we have $\texttt{di}\left(f_1, f, \mathcal{I}^k\right)$ iff $f_1 \not\equiv_k f$. Therefore, the number of pairs of correct-incorrect functions from C_l which are distinguished by $\mathcal{I}^k$ is the number of classes of equivalence minus 1 (note that the elements in the class of equivalence of f_1 are not distinguishable), multiplied by the number of elements in the class of equivalence. Thus we have

$$|\{(f', f) \mid f \in E_2,\ f \in C^l \backslash \{f_1\},\ \texttt{di}\left(f, f', \mathcal{I}^k\right)\}| = \frac{s_l}{s_k} \cdot \left(\left|\{[f] \mid f \in C^l,\ f_1 \not\equiv_k f\}\right| - 1\right) = \frac{s_l}{s_k} \cdot (s_k - 1)$$

Therefore

$$\texttt{d-rate}\left(\mathcal{I}^k, C^l, E_2\right) = \frac{|\{(f', f) \mid f \in E_2,\ f \in C^l \backslash \{f_1\},\ \texttt{di}\left(f, f', \mathcal{I}^k\right)\}|}{|E_2| \cdot |C^l \backslash E_2|} = \frac{(s_k - 1) \cdot \frac{s_l}{s_k}}{s_l - 1} = \frac{s_l - \frac{s_l}{s_k}}{s_l - 1} > \frac{s_l - \frac{s_l}{s_k}}{s_l} = 1 - \frac{1}{s_k}$$

Thus, $\texttt{d-rate}\left(\mathcal{I}^k, C^l, E_2\right)$ is higher than $1 - \frac{1}{s_k}$. Note that this lower bound does not depend on l. Thus, if we want to find $\mathcal{I}$ and n such that $\texttt{d-rate}\left(\mathcal{I}, C^l, E_2\right)$ is at least ε for all $l > n$, then we just have pick any value of n such that $1 - \frac{1}{s_n} > \varepsilon$ (that is, $1 - \varepsilon > \frac{1}{s_n}$) and pick $\mathcal{I} = I^n$. Note that s_n strictly grows with n, so such n exists. We conclude $(C_2, E_2) \in \texttt{Class II}$. □

The next example shows that some reduction of the set C_2 of the previous example makes us *lose* the inclusion in `Class II`.

Example 5: Let us reconsider (C_2, E_2) as defined in Example 4. We modify C_2 so that its pair with E_2 no longer belongs to `Class II`. Let $C_2' \subset C_2$ consist of f_1 as well as all functions g_k, with $k \in \mathbb{N}$, such that g_k behaves as f_1 for all input sequences but the input sequence $a^k b$ (and its extensions $a^k b\sigma$, for all $\sigma \in \{a, b\}^*$). In particular, the output produced by g_k when input b is given after a^k is the opposite as the one given by f_1 (i.e. 0 if it is 1 in f_1; 1 otherwise). For all input sequence $a^k b\sigma$, the outputs produced by g_k for the rest of inputs after $a^k b$ are the same as the ones produced by f_1. Therefore, for any non-decreasing sequence: $C^1 \subseteq C^2 \subseteq \cdots C^n \subseteq C^{n+1} \subseteq \cdots$ of subsets of C_2', the ratio $\texttt{d-rate}\left(\mathcal{I}, C^n, E^n\right)$ is arbitrarily low as n increases. This is because, for all finite test suite $\mathcal{I}$, the number of pairs of $\{(f_1, f) | f \in C_2' \backslash \{f_1\}\}$ distinguished by $\mathcal{I}$ is finite (in particular, this number is lower than or equal to $|\mathcal{I}|$ because, if some $i \in \mathcal{I}$ distinguishes f_1 from some wrong function f', then it cannot distinguish f_1 from any other wrong function). As a consequence, no sequence of subsets like that required by Definition 7 can be constructed. Therefore, $(C_2', E_2) \notin \texttt{Class II}$. □

The two cases considered in the two previous examples illustrate a fact that could look very surprising at a first glance: By *reducing* the computation formalism from C_2 to $C_2' \subset C_2$ (and thus reducing the number of possible *wrong* implementations) the unboundedly-approachable finite testability is *lost*. Intuitively, the reason is that C_2' only includes functions with a single fault, while *all* FSMs are in C_2 (including FSMs strongly deviating from the correct behavior). Thus, distinguishing correct implementations from incorrect ones is easier in C_2 than in C_2'. This shows that the difficulty of testing does not lie in the number of possible wrong implementations to be discarded, but in the *narrowness* of the border between correct and incorrect potential implementations. Due to this narrowness, if the computation formalism is C_2' then testing is not very productive in terms of distinguishability: After applying n tests, no more than n pairs of correct/incorrect functions will be distinguished – out of an infinite number of pairs of correct/incorrect functions to be distinguished.

Two additional elaborated examples concerning `Class II` are given in Section 5. In Case Study 5.1 we show how the classical framework of M. Hennessy [17] for process algebras can be expressed into our formalism, and we show its relation with `Class II`. In Case Study 5.2 we present a variant of FSMs where states are also endowed with timeout transitions. Timeouts of states belong to a dense domain and can be arbitrarily close to 0, which implicitly eliminates the possibility to discretize the times when we test the IUT in order to reach completeness. Yet we prove that this case belongs to `Class II`.

Let us note that `Class I` serves us a reference ideal class of what we would like to achieve by testing: precisely deciding whether the IUT is right or wrong. Although finite complete testability is not a very likely scenario in practice, knowing `Class I` (and the properties that allow to be in it) lets us assess other more realistic testing cases which are not in `Class I`: In general, we will be willing to be as *close* to `Class I` (i.e. as near to fulfill the properties required to be in the class) as possible. Later we will consider several formal ways to approach `Class I` such as e.g. assuming hypotheses regarding which functions can be in the set C (Section 4.3). However, note that `Class II` introduces by itself a practical *approximation* to `Class I`. If we are in `Class II`, then any partial distinguishing rate < 1 can be reached by some finite test suite. Note that this notion gives testers a formal measure of the *productivity* of testing. The tester intuition says that the usefulness of testing decays as more tests are applied, that is, after applying n test cases, the *utility* of applying one more test (i.e. the capability of detecting a fault if none was detected so far) decreases with n. However, how much? The construction required by Definition 7 for being in `Class II` implicitly lets us calculate that measure, because we can use it to define an expression denoting, in each particular case, the relation between the distinguishing rate and the size of test suites reaching that rate. For instance, a simple *productivity* metric, denoting the marginal utility of applying one more test after applying n test cases, could consist in the *derivative* of the distinguishing rate with respect to the size of the test suites reaching that rate. Though the distinguishing rate expressions developed before in Example 4, and later in case studies 5.1 and 5.2, implicitly let us calculate these relations, explicitly investigating them (as well as possible separations of `Class II` into subclasses in terms of the kinds of these relations) is out of the scope of this paper and will be investigated in our future work. This issue will be revisited later in Section 7, in the conclusions.

4 Studying properties of Class I

In this section we study the properties of `Class I`. We show some conditions required for finite testability, and

we present some alternative characterizations. Transformations keeping the testability are defined, and the effect of adding testing hypotheses is analyzed. The complexity of finding minimum complete test suites is studied, and we also measure how far a test suite is from being complete. Besides, we study how to reduce a *testing problem* into another, thus allowing us to find complete test suites for the former in terms of the latter. Some properties can be applied to other testability classes as well, though a specific study of the remaining four classes is left as future work. For the sake of readability, in results presented from now on we will assume that C denotes a computation formalism for a set of inputs I and a set of distinguished outputs O, and $E \subseteq C$ is a specification.

4.1 Characterizations of `Class I`

Next we present some simple conditions enabling/disabling the testability.

Lemma 1: We have the following properties:

(a) Let $f \in E$, $f' \in C \backslash E$ be such that for all $i \in I$ we have $o_1 \stackrel{\circ}{=} o_2$ for some $o_1 \in f(i)$, $o_2 \in f'(i)$. Then, $(C, E) \notin$ `Class IV`.

(b) $(C, E) \notin$ `Class I` iff for all $n \in \mathbb{N}$ and $\mathcal{I} = \{i_1, \ldots, i_n\} \subseteq I$ there exist $f \in E$, $f' \in C \backslash E$ such that for all $i \in \mathcal{I}$ we have $o_1 \stackrel{\circ}{=} o_2$ for some $o_1 \in f(i)$, $o_2 \in f'(i)$.

(c) Let us suppose that C is a *finite* computation formalism (that is, C is a finite set), and there do not exist $f \in E$, $f' \in C \backslash E$ such that for all $i \in I$ we have $o_1 \stackrel{\circ}{=} o_2$ for some $o_1 \in f(i)$, $o_2 \in f'(i)$. Then, $(C, E) \in$ `Class I`.

(d) Let us suppose that I is infinite and for some $f \in E$ we have that, for all $i \in I$, there exists $f' \in C \backslash E$ such that $f(i) \stackrel{\circ}{\neq} f'(i)$, but $f(i') \stackrel{\circ}{=} f'(i')$ for all $i' \neq i$ with $i' \in I$. Then, $(C, E) \notin$ `Class I`.

□

Proof. Properties (1) and (2) are straightforward. Next we prove (3). Let us consider the set $A = \{(f, f') | f \in E \wedge f' \in C \backslash E\}$. Let us note that A is finite because C is finite. Since we assume that there do not exist $f \in E$ and $f' \in C \backslash E$ such that $f(i) \stackrel{\circ}{=} f'(i)$ for all $i \in \mathcal{I}$, for each pair $(f, f') \in A$ there exists some input $i \in I$ allowing to distinguish f and f', that is, there exists $i \in I$ such that $f(i) \stackrel{\circ}{\neq} f'(i)$. Thus, a finite set containing an input allowing to distinguish f and f' for each pair (f, f') is a finite complete test suite for C and E. Thus, $(C, E) \in$ `Class I`.

Now we prove (4). Let us consider a finite test suite $\mathcal{I} \subseteq I$. Since $\mathcal{I}$ is finite, there is $i \in I$ such that $i \notin \mathcal{I}$. By hypothesis, there is $f' \in C \backslash E$ such that $f'(i) \stackrel{\circ}{\neq} f(i)$ but $f(i') \stackrel{\circ}{=} f'(i')$ for all $i' \in I$ with $i \neq i'$. Since $i \notin \mathcal{I}$, $\mathcal{I}$ does not distinguish f and f'. Thus, $\mathcal{I}$ cannot be complete. Since this happens for any finite test suite, we conclude $(C, E) \notin$ `Class I`. □

The previous results are, in fact, a simple rephrasing of definitions given in Section 2. However, they help us to reason about the testability of problems from an abstract point of view, in such a way that some (or most of) details concerning the structure of a computation formalism (states, instructions, etc) can be ignored. The next example illustrates how the previous results can be applied.

Example 6: Let (C_2, E_2) be defined as in Example 3. We argued that, in general, $(C_2, E_2) \notin$ `Class I`. Now we prove it by applying Lemma 1 (b). Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be any finite set of sequences of I' inputs, and let k be the length of the longest sequence in $\mathcal{I}$. Let $E_2 = \{f\}$ where f denotes the behavior of a deterministic completely-specified FSM M. Since the number of states of FSMs represented by C_2 is not bounded, there exists $f' \in C_2$ representing the behavior of an FSM M' such that for all $i \in \mathcal{I}$ the response is the same as the one given by M, but the answer is different for some sequence of length $k+1$. Hence, $f \in E_2$ and $f' \notin E_2$ are not distinguished by $\mathcal{I}$. By Lemma 1 (b), $(C_2, E_2) \notin$ `Class I`.

Let (C_1, E_1) be defined as in Example 3. We provide a simple proof of $(C_1, E_1) \in$ `Class I` which does not require to reason about the set of all traces of some length. Since we consider deterministic FSMs, for all $f_1, f_2 \in C_1$ there exists $i \in I$ distinguishing f_1 and f_2. Thus, for all $f \in E_1$ and $f' \in C_1 \backslash E_1$ there exists some $i \in I$ distinguishing f and f'. Since C_1 is *finite*, by Lemma 1 (c) we conclude $(C_1, E_1) \in$ `Class I`. Let us apply Lemma 1 (c) in a similar way, but in a very different context. Let P denote the set of all deterministic programs written in some programming language with less than k characters, I denote the (finite) set of *keys* in a standard keyboard, and O denote the (finite) set of *ASCII symbols* with the trivial distinguishing relation, respectively. Let C_1' represent the *beginning* of the behavior of all programs in P as follows: If $f \in C_1'$ represents $p \in P$ then $f(i) = \{o\}$ denotes that $o \in O$ is the first character depicted in the screen by p if a keystroke $i \in I$ is produced at the *first* execution tick. If no character is depicted after t ticks then we consider $f(i) = \{Null\}$ with $Null \in O$. Let $E_1' = \{f\}$ for some $f \in C_1'$. Since C_1' is finite and all functions in C_1' are distinguishable, again by Lemma 1 (c) we have $(C_1', E_1') \in$ `Class I`.

Lemma 1 (c) provides some sufficient conditions to achieve finite testability, though they are not necessary. We may also have finite testability when the computation formalism is infinite. The finitely testable pair (C, E) considered in the last paragraph of Example 3 in Section 3, dealing with an *infinite* set of programs of some kind, illustrated this fact. Next we consider an example in the context of FSMs. Let C_6 represent all deterministic FSMs and $E_6 \subseteq C_6$ represent all deterministic FSMs with $I' = \{a, b\}$ such that, if they receive a in their initial state, then they produce $q \in O'$, and any behavior is allowed next. This means that, for all sequence of I' inputs beginning by a, the sequence of O' outputs must begin by q. We are imposing a requirement over all sequences beginning by a, that is $a, aa, ab, aaa, aab, aba, abb, \ldots \in I$. However, we do not have to *test* all sequences in this infinite set. In fact, $\{a\}$ is a complete test suite for (C_6, E_6) because, for all $f \in C_6$ (where f may or may not belong to E_6), we have the following property: If $f(a) = \{q\}$

then for all w we have $f(aw) = \{qw'\}$ for some w' (otherwise, f does not represent a *deterministic* FSM, which is required by C_6). Thus, the input a distinguishes any pair of functions $f \in E_6$ and $f' \in C_6 \backslash E_6$. Hence, $\{a\}$ is a (finite) complete test suite for the *infinite* sets C_6 and E_6, and we have $(C_6, E_6) \in \texttt{Class I}$.

Let us consider again programs written in some programming language. Let P denote the set of all *terminating* deterministic programs such that, after launching them, they wait for receiving a character sequence from the user and next they display a sequence of characters. Let us note that this time we are not constraining the *length* of programs, so the source code of these programs may have any length. Let $I = O$ denote the set of all sequences of ASCII characters. Let C_1'' represent all programs in P as follows: If $f \in C_1''$ represents $p \in P$ then $f(i) = \{o\}$ denotes that the sequence of ASCII symbols $o \in O$ is depicted by p if, after launching p, the user writes the sequence $i \in I$ and next presses *enter*. If no sequence is depicted before the termination of p then we consider $f(i) = \{Null\}$ with $Null \in O$. Let $E_1'' = \{f\}$ consist of a single function $f \in C_1''$. Given any deterministic program p and a sequence of characters i, we can trivially construct another deterministic program p' such that p' behaves as p for all sequences of characters but i (we just catch this special case with an *if-then-else* statement at the beginning). Hence, for all $i \in I$ we can find a function $f' \in C_1'' \backslash E_1''$ such that f and f' are only distinguished by i. Since the set of all sequences of ASCII characters is infinite, by Lemma 1 (d) we conclude $(C_1'', E_1'') \notin \texttt{Class I}$.

Let us revisit (C_6, E_6) given before in this example and let $f \in E_6$ be an arbitrary function. We cannot find, for all $i \in I$, a function $f' \in C_6 \backslash E_6$ such that f and f' are only distinguished by i, because f and f' must answer different outputs for *infinite* inputs (in particular, for all sequences beginning by a). Thus, in this case we *cannot* apply Lemma 1 (d) to infer $(C_6, E_6) \notin \texttt{Class I}$. □

Before presenting the next results, let us introduce some notation to facilitate the reading. Next we identify the set of all inputs that distinguish two functions.

Definition 8: Let $f \in E$ and $f' \in C \backslash E$. We define the *set of inputs that distinguish f and f'*, written $\texttt{disobs}(f, f')$, as:

$$\texttt{disobs}(f, f') = \{i \mid i \in I \ \wedge \ f(i) \not\supseteq f'(i)\}$$

□

The next result analyzes conditions preserving testability, that is, we study how we can modify C or E in such a way that complete test suites are preserved. Let us suppose that distinguishing $f_1 \in E$ and $f_1' \in C \backslash E$ requires applying an input that, in particular, also distinguishes $f_2 \in E$ and $f_2' \in C \backslash E$. Then test suites do not need to care about f_2 and f_2' in order to achieve completeness, so we can ignore them. In this way, smaller sets of functions can be considered.

Lemma 2: *(Equi-testability Lemma)* Let C be a computation formalism for I and O, and let $E \subseteq C$. Let $C_0 \subseteq C$ be such that for all $f \in E$ and $f' \in C \backslash E$ there exist $g \in E \backslash C_0$ and $g' \in (C \backslash E) \backslash C_0$ verifying $\texttt{disobs}(g, g') \neq \varnothing$ and $\texttt{disobs}(g, g') \subseteq \texttt{disobs}(f, f')$. Then, $\mathcal{I}$ is a complete test suite for C and E iff $\mathcal{I}$ is a complete test suite for $C \backslash C_0$ and $E \backslash C_0$. □

Proof.

$\Longrightarrow$: Let $\mathcal{I}$ be a complete test suite for C and E. Since $E \backslash C_0 \subseteq E$ and $(C \backslash E) \backslash C_0 \subseteq C \backslash E$, $\mathcal{I}$ is a complete test suite for $C \backslash C_0$ and $E \backslash C_0$.

$\Longleftarrow$: Let $\mathcal{I}$ be a complete test suite for $C \backslash C_0$ and $E \backslash C_0$. Let us prove that $\mathcal{I}$ is a complete set for C and E. Let us consider $f \in E$ and $f' \in C \backslash E$. By hypothesis, there exists $g \in E \backslash C_0$ and $g' \in (C \backslash E) \backslash C_0$ such that $\texttt{disobs}(g, g') \neq \varnothing$ and $\texttt{disobs}(g, g') \subseteq \texttt{disobs}(f, f')$. Since $\mathcal{I}$ is complete for $C \backslash C_0$ and $E \backslash C_0$, there is $i \in \mathcal{I}$ such that $g(i) \not\supseteq g'(i)$. By the definition of $\texttt{disobs}(g, g')$, we have $i \in \texttt{disobs}(g, g')$. Since $\texttt{disobs}(g, g') \subseteq \texttt{disobs}(f, f')$, again by the definition of $\texttt{disobs}(f, f')$ we conclude $f(i) \not\supseteq f'(i)$. Thus, $\mathcal{I}$ is a complete test suite for (C, E).

□

Example 7: We apply Lemma 2 to find out that we can ignore some functions in the search for a complete test suite. Let C consist of some non-deterministic functions for input set I and output set O, and let $E = \{f\}$ for some $f \in C$. For some $i \in I$, let V be the set of all functions $f' \in C \backslash E$ such that $f(i)$ and $f'(i)$ are disjoint, and let $g \in V$ be some function such that, for all $i' \in I \backslash \{i\}$, we have $g(i') = f(i')$. Note that for all $f' \in V$ we have $\texttt{disobs}(f, g) = \{i\} \subseteq \texttt{disobs}(f, f')$. Thus, by Lemma 2 we conclude that a set $\mathcal{I} \subseteq I$ is a complete test suite for (C, E) iff $\mathcal{I}$ is a complete test suite for $((C \backslash V) \cup \{g\}, E)$. Hence, all functions in V but g can be ignored, and just the smaller set $(C \backslash V) \cup \{g\} \subseteq C$ has to be considered to seek for the completeness for C. Actually, $((C \backslash V) \cup \{g\}, E) \in \texttt{Class I}$ would imply $(C, E) \in \texttt{Class I}$. □

A very different approach to reduce a testing problem into another will be presented later in Definition 12.

Next we present a result providing a lower bound of the number of inputs that must be included in a set to achieve a complete test suite.

Proposition 1: Let $\mathcal{A} \subseteq 2^I$ be a pairwise disjoint[6] set of *sets of inputs* such that for all $A \in \mathcal{A}$ there exist $f \in E$, $f' \in C \backslash E$ verifying $\texttt{disobs}(f, f') \subseteq A$. Then $|\mathcal{I}| \geq |\mathcal{A}|$ for any finite complete test suite $\mathcal{I}$ for C and E. □

Proof. Let us assume that there exists a complete test suite $\mathcal{I} \subseteq I$ for (C, E). In order to prove the result, we define an injective function $h : \mathcal{A} \mapsto \mathcal{I}$. For any $A \in \mathcal{A}$, let us take $f \in E$ and $f' \in C \backslash E$ fulfilling the premises of the proposition. Since $\mathcal{I}$ is complete, there must be some $i_A \in \mathcal{I}$ satisfying

$$i_A \in \mathcal{I} \cap \{i \mid i \in I \ \wedge \ f(i) \not\supseteq f'(i)\} \subseteq \mathcal{I} \cap A$$

Let us define $h(A) = i_A$. This function is injective since $\mathcal{A}$ is pairwise disjoint. Thus, $|\mathcal{I}| \leq |\mathcal{A}|$. □

6. That is, for all $A, A' \in \mathcal{A}$ we have $A = A'$ or $A \cap A' = \varnothing$.

Let us note that if $\mathcal{A}$ is not finite then, by applying the previous proposition, we immediately infer $(C, E) \notin$ `Class I`. The next example uses this property.

Example 8: Let (C_2, E_2) be defined as in Example 3. In particular, $E_2 = \{f\}$ consists of a single correct function, $I = I'^*$ denotes the set of all sequences of I' inputs, and $a, b \in I'$ denote any two inputs. Since C_2 does not limit the number of states of represented FSMs, for all sequence $i \in I'^*$ we can find two FSMs represented by C_2 whose answers differ only for i (as well as its extensions $i\sigma$ for all $\sigma \in I'^*$). Let $Q_k = \{a^k b\sigma | \sigma \in I'^*\}$. For all $k \in \mathbb{N}$ we can find a function $f' \in C\backslash E$ such that *only* those input sequences that are included in Q_k allow to distinguish f and f' (in particular, this is so if f' behaves as f for all input sequences but $a^k b$, where a wrong output is produced). For all $k_1, k_2 \in \mathbb{N}$ with $k_1 \neq k_2$ we have $Q_{k_1} \cap Q_{k_2} = \varnothing$. Thus, the infinite set $\mathcal{A} = \{Q_k | k \in \mathbb{N}\}$ fulfills the conditions of Proposition 1. Hence, we rediscover $(C_2, E_2) \notin$ `Class I`.

Interestingly, we may have $(C, E) \notin$ `Class I` even if there does not exist any infinite set $\mathcal{A}$ fulfilling the conditions of Proposition 1 (that is, the longest set fulfilling the conditions is finite). Let us show it by explicitly constructing such a case. Let C be a computation formalism and $E \subseteq C$ be a specification such that C and $C\backslash E$ are *infinite* sets. Each input will be decorated with the name of two correct and two incorrect *functions*. Specifically, $I = \left\{ i_{x,y}^{x',y'} \,|x, y \in E \ \wedge \ x', y' \in C\backslash E \right\}$. Besides, we consider $O = \{is_x | x \in C\} \cup \{notdist_{x,y} | x, y \in C\}$. As we will see, an output of the form is_f will identify the function f emitting it, because it will be emitted only by f. Besides, an output of the form $notdist_{x,y}$ will make functions x and y not to be distinguished by the input leading to emit that output, because both x and y will emit it. We will assume that the ordering of indexes in inputs and outputs is irrelevant, so e.g. we assume $i_{f,g}^{f',g'} = i_{f,g}^{g',f'}$ and $notdist_{h,h'} = notdist_{h',h}$. Note that I and O are infinite sets.

For all function $q \in C$ we define $q\left(i_{f,g}^{f',g'}\right) = \{is_q\} \cup \{notdist_{q,h} | h \in C\backslash\{f, f', g, g'\}\}$ if $q \in \{f, f', g, g'\}$; else $q\left(i_{f,g}^{f',g'}\right) = O$. According to this construction, input $i_{f,g}^{f',g'}$ distinguishes h and h' if $h \in \{f, g\}$ and $h' \in \{f', g'\}$ because we have $h\left(i_{f,g}^{f',g'}\right) \cap h'\left(i_{f,g}^{f',g'}\right) = \varnothing$. Otherwise, i.e. if $h \notin \{f, g\}$ or $h' \notin \{f', g'\}$, we have $h\left(i_{f,g}^{f',g'}\right) \cap h'\left(i_{f,g}^{f',g'}\right) \neq \varnothing$, so $i_{f,g}^{f',g'}$ does not distinguish h and h'. Let W be the condition, required by Proposition 1, that for all $A \in \mathcal{A}$ there exist $f \in E$, $f' \in C\backslash E$ verifying $\{i \mid i \in I \ \wedge \ f(i) \neq f'(i)\} \subseteq A$. Any set of inputs A fulfilling condition W must include, for some pair of functions f, f', all inputs allowing to distinguish f, f'. For any other set A' including all inputs allowing to distinguish another pair g, g', the set A shares at least one input with A' (in particular, A and A' share the input $i_{f,g}^{f',g'}$). Thus, there do not exist two disjoint sets A and A' fulfilling condition W. Since Proposition 1 also requires that all sets of $\mathcal{A}$ are pairwise disjoint, the conditions of the proposition are fulfilled only if $\mathcal{A} = \{I\}$, i.e. $\mathcal{A}$ cannot be infinite. However, there does not exist any finite complete test suite for C and E either: Each input $i_{f,g}^{f',g'}$ distinguishes only four pairs (in particular, any combination of f or f' with g or g'), but there are infinite pairs to be distinguished (recall that we are assuming that E and $C\backslash E$ are infinite sets). Thus, $(C, E) \notin$ `Class I`. □

Next we consider an alternative way to find complete test suites by manipulating the sets distinguishing each pair of functions.

Proposition 2: Let G be the set of all *sets of inputs* allowing to distinguish each correct function from each incorrect function, that is:

$$G = \left\{ \texttt{disobs}(f, f') \mid f \in E \ \wedge \ f' \in C\backslash E \right\}$$

We have $(C, E) \in$ `Class I` iff there exist $n \in \mathbb{N}$ subsets of G, denoted by $A_1, \ldots, A_n \subseteq G$, such that $G = \bigcup_{1 \leq j \leq n} A_i$ and $\bigcap_{B \in A_j} B \neq \varnothing$ for all $1 \leq j \leq n$.

Moreover, if such n subsets of G exist, then there is a complete test suite $\mathcal{I}$ such that $|\mathcal{I}| \leq n$. □

Proof.

$\Longrightarrow$: If $(C, E) \in$ `Class I` then there exists a finite complete test suite $\{i_1, \ldots, i_n\}$ for C and E. For each i_j, let A_j be the set of all sets of inputs distinguishing a correct and an incorrect function such that i_j is included, that is, $A_j = \{B \mid B \in G \ \wedge \ i_j \in B\} \subseteq G$. We have $i_j \in \bigcap_{B \in A_j} B$, so $\bigcap_{B \in A_j} B \neq \varnothing$. For all $B \in G$, we have $B \cap \{i_1, \ldots, i_n\} \neq \varnothing$ (otherwise, some pair of functions would not be distinguished by $\{i_1, \ldots, i_n\}$ and it would not be a complete test suite). So, for some $1 \leq j \leq n$ we have $i_j \in B \cap \{i_1, \ldots, i_n\}$. Thus, by the construction of sets A_j, for all $B \in G$ we have $B \in A_j$ for some $1 \leq j \leq n$, and we conclude $\bigcup_{1 \leq j \leq n} A_j = G$.

$\Longleftarrow$: This implication is easy to prove: We can build a finite complete test suite by taking, for all $1 \leq j \leq n$, any input from the set $\bigcap_{B \in A_j} B \neq \varnothing$. Let us note that the size of the resulting test suite is n.

□

In the following example we show an application of the previous property.

Example 9: Let us consider $I = \mathbb{N}$, $O = \{0, 1\}$, and for all $i \in \mathbb{N}^+$ the functions $f_i(x) : \mathbb{N} \mapsto 2^{\{0,1\}}$ defined as:

$$f_i(x) = \begin{cases} \{1\} & \text{if } x \mod i = 0 \\ \{0\} & \text{if } x \mod i \neq 0 \end{cases}$$

Let $C = \{f_i \mid i \in \mathbb{N}^+\}$ and $E = \{f_6\}$. Let us consider $B_i = \texttt{disobs}(f_i, f_6)$ for $i > 0$, $i \neq 6$. We check that the sets $A_1 = \{B_1\}$, $A_2 = \{B_2\}$, $A_3 = \{B_3\}$ and $A_4 = \left\{B_i \mid i > 0 \wedge i \notin \{1, 2, 3, 6\}\right\}$ satisfy the hypothesis of Proposition 2. First we check that $G = A_1 \cup A_2 \cup A_3 \cup A_4$. This is obvious because $G = \{B_i \mid i > 0 \ \wedge \ i \neq 6\}$. Now let us check that $\bigcap_{B \in A_i} B \neq \varnothing$ for $i \in \{1, 2, 3, 4\}$. Since $|A_i| = 1$ for $i \leq 3$, this condition is trivial for these sets. Finally, we have $6 \in B_i$ for $i \notin \{1, 2, 3, 6\}$, so $\bigcap_{B \in A_4} B \neq \varnothing$.

Thus, by Proposition 2 we conclude $(C, E) \in$ `Class I`. Moreover, we can easily extract a complete test suite from sets A_1, A_2, A_3, A_4: In order to distinguish all pairs of functions (f_i, f_6), for each set A_i we just have to pick any element from the intersection of all sets in A_i. In fact, $\{1, 2, 3, 6\}$ is a complete test suite for C and E. □

4.2 Minimum Test Suites

Next we consider the problem of finding the *minimum* complete test suite in a particularly basic case:

(a) the computation formalism $C = \{f_1, \ldots, f_n\}$ is finite,
(b) the sets $I = \{i_1, \ldots, i_k\}$ and $O = \{o_1, \ldots, o_l\}$ of inputs and outputs are finite as well.

Before continuing, let us note that any function $f : I \mapsto 2^O$ can be considered as a set of pairs $f \subseteq I \times 2^O$ such that for all $i \in I$ there is a single $outs \in 2^O$ with $(i, outs) \in f$. Therefore, any finite function can be extensionally defined as the set of tuples $f = \{(i_1, outs_1), \ldots (i_k, outs_k)\}$. Since we have already enumerated the set of inputs $I = \{i_1, \ldots, i_k\}$, the function f can be implicitly defined by the tuple $(outs_1, \ldots, outs_k)$. Thus, in the rest of the paper we will identify any finite function $f \in C$ with such a tuple: $f = (outs_1, \ldots, outs_k)$.

We will call the problem of finding a complete test suite of some size or smaller under the previous conditions the *Complete Suite* problem, and we will denote it by `CS`. So, the corresponding optimization problem consists in finding the minimum complete test suite.

Definition 9: Let C be a finite computation formalism for the finite set of inputs and outputs I, a set of outputs O with a distinguishing relation $\not\equiv$, and $E \subseteq C$ be a specification.

Given C, E, I, O, $\not\equiv$, and some $K \in \mathbb{N}$, the *Complete Suite* problem (`CS`) is defined as follows: Is there any complete test suite $\mathcal{I}$ for C and E such that $|\mathcal{I}| \leq K$? □

Theorem 1: `CS` $\in$ `NP-complete`. □

The proof of this theorem is given in Section 6. Proving `CS` $\in$ `NP` is straightforward, whereas `CS` $\in$ `NP-hard` is proved by reducing the *Set Cover* problem to `CS`.

This result shows us that, in general, finding the minimum complete test suite (if it exists) is a hard problem, as the NP-completeness of the decision problem implies the NP-hardness of the corresponding optimization problem. We immediately infer that other useful generalizations of this optimization problem are also NP-hard. For instance, finding the minimum test suite reaching some given *distinguishing rate* $d \leq 1$ (see Definition 6); finding the test suite which maximizes some linear combination of the distinguishing rate and (the inverse of) the number of test cases in the suite; or doing that under the assumption test cases might have different *costs* and these costs are explicitly given, generalize that problem, so they are trivially NP-hard problems as well.

Several testing techniques face the problem of measuring the (a priori) capability of test suites to find faults (e.g. Mutation Testing [11], [21]), and thus they also support methodologies to select good test suites: We pick those test suites reaching the highest score according to some fault-detection capability measure. Theorem 1 shows us that these tasks will be hard in general. Note that, although `CS` is defined in terms of our completeness criterion (i.e. distinguishing all pairs of correct/incorrect functions), `CS` would remain NP-complete even if we assumed the more particular case where there is a single acceptable behavior and all other possible behaviors are incorrect.[7] In this case, we would not speak about distinguishing correct/incorrect pairs of functions, but just about *discarding faulty implementations*, which actually fits into other test-suite fault-detection metrics used in the literature.[8] However, the problem would still be NP-complete in this particular case. Of course, the problem may be polynomial if only some particular kind of systems is considered.

4.3 Using testing hypotheses to enable finite testability

In this section we present some notions allowing us to reason about *testing hypotheses*, which play a key role in formal testing methodologies. Assuming a testing hypothesis typically consists in making an assumption about the set of systems that could actually be the IUT. As we have seen in several examples, the testability is affected by assuming some restrictions about the IUT (e.g. testing deterministic FSMs is not in `Class I`, but it is if a given limit of the number of states is assumed). Thus, assuming a testing hypothesis might allow an incomplete test suite $\mathcal{I}$ to become complete (provided that the hypothesis actually holds). Next we study the conditions for this. We will denote a hypothesis by the set H of systems we *remove* from those that could actually be the IUT if the hypothesis is assumed.

Definition 10: A *testing hypothesis* H for C is a subset $H \subseteq C$.

Let $\mathcal{I} \subseteq I$ *not* be a complete test suite for C and E. If $\mathcal{I}$ is a complete test suite for $C \backslash H$ and $E \backslash H$, then we say that H *enables* $\mathcal{I}$ for (C, E).

Let $(C, E) \notin$ `Class I`. If $(C \backslash H, E \backslash H) \in$ `Class I` then we say that H *enables* (C, E). □

Lemma 3: Let $\mathcal{I} \subseteq I$ be a finite set of inputs that is *not* a complete test suite for C and E and let H be a testing hypothesis for C. H *enables* $\mathcal{I}$ for (C, E) iff, for all $f \in E$

7. In the proof of NP-completeness of `CS` given in Section 6, the polynomial reduction used to prove its NP-hardness considers a single *incorrect* function. Since the problem treats correct and incorrect functions symmetrically, that reduction also works if correct and incorrect function roles are trivially exchanged so that a single correct function is considered.

8. For instance, in the simplest form of Mutation Testing, a single program behavior is considered correct, which is the behavior of the specification program (obviously, it is also the behavior of any other equivalent program). All other behaviors are considered incorrect. Then, a measure of the quality of a test suite is the portion of incorrect functions (behaviors) detected by the suite (i.e. the number of incorrect *mutants* it *kills*).

and $f' \in C\backslash E$ such that $f(i) \hat{=} f'(i)$ for all $i \in \mathcal{I}$, either $f \in H$ or $f' \in H$ holds. □

Proof.

$\Longrightarrow$: Let us consider $f \in C$ and $f' \in C\backslash E$ such that for all $i \in \mathcal{I}$ we have $f(i) \hat{=} f'(i)$. Since $\mathcal{I}$ is a complete test suite for $(C\backslash H, E\backslash H)$, we infer $f \notin C\backslash H$ or $f' \notin E\backslash H$. So $f \in H$ or $f' \in H$.

$\Longleftarrow$: In order to check that $\mathcal{I}$ is complete for $(C\backslash H, E\backslash H)$, let us consider $f \in E\backslash H$ and $f' \in (C\backslash H)\backslash(E\backslash H) = (C\backslash E)\backslash H$. Since $f \notin H$ and $f' \notin H$, there is $i \in \mathcal{I}$ such that $f(i) \not\hat{=} f'(i)$. Then $\mathcal{I}$ is complete for $(C\backslash H, E\backslash H)$.

□

In Section 3.2 we proposed to use the *distinguishing rate* (see Definition 7) to measure the coverage of an incomplete finite test suite. Alternatively, next we consider measuring the coverage of an incomplete test suite $\mathcal{I}$ in terms of the amount of potential functions that should be *removed* from C to make $\mathcal{I}$ *complete*, that is, in terms of the number of potential implementations we have to *assume* not to be the actual IUT to make $\mathcal{I}$ complete. By removing functions from C, the number of pairs of complete/incomplete functions to be distinguished is reduced. This may turn an incomplete test suite into complete or, moreover, enable finite testability in a problem that did not belong to `Class I` before removing functions.

According to this idea, we consider that an incomplete test suite is *better* than another incomplete suite if making the former suite complete requires removing less functions. That is, the suite requiring a kind of *weaker* function removal assumption to be complete is better. As we have said before, testing hypotheses are assumed to make this effect indeed, that is, to reduce the number of potential implementations so that finite test suites can be complete – provided that the hypotheses hold. Thus, if a test suite requires less or weaker hypotheses to be assumed (i.e. less potential implementations have to be removed) to get completeness than another, then the former test suite is better because it is *closer* to be complete indeed. The next example illustrates this idea.

Example 10: Let us suppose that there are only two correct functions f_1 and f_2 in E, and only two incorrect functions f_1' and f_2' in $C\backslash E$. Let us suppose that test suite $\mathcal{I}_1$ does not distinguish f_2 from f_1' or f_2', but it distinguishes all remaining pairs of correct and incorrect functions. Besides, $\mathcal{I}_2$ does not distinguish f_1 from f_1' nor f_2 from f_2', but it distinguishes all other correct-incorrect pairs. Note that the number of pairs not distinguished by each test suite is 2 in both cases, so the *distinguishing rate* (see Definition 6) would not allow us to prefer $\mathcal{I}_1$ over $\mathcal{I}_2$ or the other way around. Besides, let us assume that

- The system represented by f_1 is deterministic; it always handles some user's request Q in the same way (that is, the way it handles Q is the same in all system configurations); and it implements correctly some functionality Z.
- f_2 is non-deterministic; it does not always handle Q in the same way (i.e. the way it handles Q is different at different system configurations); and it implements correctly functionality Z.
- f_1' is deterministic; it always handles Q in the same way; and it implements incorrectly functionality Z.
- f_2' is deterministic; it does not always handle Q in the same way; and it implements correctly functionality Z.

Let us suppose that the tester can assume (or not) the following testing hypotheses: (i) the IUT is deterministic; (ii) request Q is always handled in the same way; (iii) functionality Z is correctly implemented. Besides, the tester assumes that the feasibility of each of these hypotheses is very similar. Which test suite ($\mathcal{I}_1$ or $\mathcal{I}_2$) requires the smallest set of hypotheses to be complete? We can see that $\mathcal{I}_1$ would become complete if hypothesis (i) were assumed: If f_2 can be discarded, then both pairs not distinguished by $\mathcal{I}_1$ (i.e. (f_2, f_1') and (f_2, f_2')) no longer exist, so $\mathcal{I}_1$ becomes a complete test suite. On the other hand, $\mathcal{I}_2$ would become complete if hypotheses (i) and (iii) were assumed, as they would let us discard f_2 and f_1', respectively (so all pairs not distinguished by $\mathcal{I}_2$ would not longer exist and $\mathcal{I}_2$ would be complete). The test suite $\mathcal{I}_2$ would also become complete if hypotheses (ii) and (iii) were assumed, but there is no hypothesis which achieves that alone. Thus, $\mathcal{I}_1$ requires a smaller set of hypotheses than $\mathcal{I}_2$ to become complete. If we can assume that all three hypotheses have the same feasibility and all are independent from each other, then we can consider that $\mathcal{I}_1$ is a *better* test suite than $\mathcal{I}_2$, because it is *closer* to be complete. □

Next we study the difficulty to assess the coverage measure of a test suite in these terms, and we do it in the same context as when we defined the `CS` problem in Definition 9, that is, C, E, I, O are finite. In the next definition, the set $\mathcal{H} = \{H_1, \ldots, H_n\}$ represents the hypotheses the tester may or may not assume: If the hypothesis $H_i \subseteq C$ is assumed then all functions in H_i are assumed not to be in the actual computation formalism (e.g. if we are assuming that the IUT is deterministic then we assume some $H \subseteq C$, where H consists of all non-deterministic functions in C).

Definition 11: Let C be a finite computation formalism for the finite set of inputs I, the finite set of outputs O with the distinguishing relation $\not\hat{=}$, and $E \subseteq C$ be a finite specification.

In addition, let $\mathcal{I} \subseteq I$ be a set of inputs and $\mathcal{H} = \{H_1, \ldots, H_n\}$, where for all $1 \leq i \leq n$ we have $H_i \subseteq C$.

Given C, E, I, O , $\not\hat{=}$, $\mathcal{I}$ and some $K \in \mathbb{N}$, the *Function Removal* problem (`FR`) is defined as follows: Is there any set $R \subseteq C$ with $|R| \leq K$ such that $\mathcal{I}$ is a complete test suite for $C\backslash R$ and $E\backslash R$?

Given C, E, I, O, $\not\hat{=}$, $\mathcal{I}$, $\mathcal{H}$, and some $K \in \mathbb{N}$, the *Function removal via Hypotheses* problem (`FH`) is defined as follows: Is there any set of hypotheses $R \subseteq \mathcal{H}$ with $|\bigcup_{H \in R} H| \leq K$ such that $\mathcal{I}$ is a complete test suite for

$C\backslash(\bigcup_{H\in R} H)$ and $E\backslash(\bigcup_{H\in R} H)$?

Given C, E, I, O, $\not=$, $\mathcal{I}$, $\mathcal{H}$, and some $K \in \mathbb{N}$, the *Hypotheses Assumption* problem (HA) is defined as follows: Is there any set $R \subseteq \mathcal{H}$ with $|R| \leq K$ such that $\mathcal{I}$ is a complete test suite for $C\backslash(\bigcup_{H\in R} H)$ and $E\backslash(\bigcup_{H\in R} H)$? □

The differences among the previous problems (and the corresponding coverage measures they allow to calculate) is the following: In FR, the coverage is measured in terms of the *number* of functions that must be removed, where *any* subset of C is allowed to be removed. In FH, the number of functions is considered again, though this time the set of functions to be removed must be the union of some *hypotheses* (sets of functions) from a given hypotheses repertory $\mathcal{H}$. Finally, HA considers the coverage in terms of the number of assumed *hypotheses*, rather than the number of removed functions. At a first glance, one might think that all FR, FH, and HA are NP-complete problems, and so their corresponding optimization problems are all NP-hard. Interestingly, FR is not NP-complete, as it can be reduced to the minimum *Vertex Cover* problem in *bipartite graphs*, which in turn is equivalent to the maximum *Matching* problem. This problem can be solved in polynomial time by the Hopcroft-Karp algorithm [20]. Thus, measuring the coverage of an incomplete test suite in terms of the minimum number of functions that must be removed to achieve completeness is a tractable problem indeed. The proof of the next result, given in Section 6, introduces the proposed reduction, and uses it to actually find the *minimum* function removal in time $\mathcal{O}(|C|^{5/2} + |C|^2 \cdot |\mathcal{I}| \cdot |O|^2)$, thus solving FR and its corresponding optimization version polynomially. On the other hand, the NP-completeness of FH and HA is proved by constructing polynomial reductions from *3-SAT* and *Set Cover*, respectively, which implies that their optimization versions are NP-hard problems.

Theorem 2: We have the following properties:

(a) FR $\in$ P
(b) FH $\in$ NP-complete
(c) HA $\in$ NP-complete

□

The proof of this result is given in Section 6.

The *distinguishing rate* notion given in Definition 6, and the previous three measures based on assessing how many (groups of) functions we should assume not to be the IUT to reach completeness, attempt to answer the question of how far an incomplete test suite is from being complete. Since they do so in an abstract manner (i.e. they do not rely on numbers of states, transitions, code lines, etc), they are general enough to be applied to any testing setting.

In particular, let us consider a setting where testing is particularly costly because e.g. each test case will take a long time, each test wastes some resources, etc. Let us suppose that a finite set of test cases is designed according to the testers' experience. Similarly, a finite fault model is constructed to denote some representative right and wrong definitions of the IUT (or just a set of its possible faults, which may be combined in the IUT in any possible way). Thus, the particular setting required in the definitions of problems CS, FR, FH, or HA (Definitions 9 and 11), which require that the sets of possible test cases and possible IUT definitions are finite and explicitly given, is met. Given a manually designed set of test cases, such that the cost of applying *all* considered test cases in the set is not feasible (due to time, money, etc), the problem of selecting a good subset from the set necessarily arises. Hence, the proposed notions of *distinguishing rate* and *least required hypothesis providing completeness* provide a general criterion to pick some of these test cases in terms of whether the set of selected test cases will have (a priori) a good capability to detect faults.

Similarly, as we said in the case of CS, we could also consider some useful variants of problems FR, FH, or HA. For instance, it is trivial to see that FH and HA would also remain NP-complete if we assigned *weights* to each function or hypothesis, respectively.[9] Note that weights would allow the tester to express that assuming some hypothesis A does not have the same likelihood than assuming another hypothesis B.

4.4 Testing reductions

In this section we present a notion to relate different testing scenarios. Given a computation formalism, we consider the problem of finding a complete test suite for *any* specification belonging to a given set of specifications. This is the typical goal of testing methodologies: For any specification fitting into a given kind of specifications (for us, a *computation formalism*), find a way to derive tests from the specification in such a way that the test suite is complete. Moreover, rather than imposing a fixed computation formalism for all possible specifications, a (possibly infinite) set of pairs (C, E) will denote the cases to be considered. This will improve the generality of the framework. For instance, if the tester assumes that the IUT includes at most n faults with respect to the specification, then a different set C should be considered for each possible set E. So, for us a *testing problem* will be a set of pairs (C, E), and the *goal* of a testing problem will be, given any pair in the set, finding a complete test suite for that pair. Ideally, our knowledge about how to find suitable tests for a given testing problem (i.e. for a given kind of target formalisms and specifications) could help us to face other testing problems. In order to enable this, we introduce a general notion of *testability reduction*. If a testing problem can be solved by *transforming* it into another testing problem and solving the latter, then we will say that the former problem can be *reduced* to the latter. This provides a criterion to classify problems inside each class.

9. The question of whether FR would still be polynomial if each function had a different weight is still open, and will be addressed in future work.

Definition 12: A *testing problem* $\mathcal{T}$ for a set of inputs I is a set $\mathcal{T}$ of pairs (C, E) with the usual meanings of C and E where, for all $(C, E) \in \mathcal{T}$, the set of inputs of C (that is, the domain of functions in C) is included in I.

Let $\mathcal{T} \subseteq$ `Class I`. A computable function $d : \mathcal{T} \longrightarrow 2^I$ is a *finite suite derivation* for $\mathcal{T}$ if, for all $p \in \mathcal{T}$, $d(p)$ is a finite complete test suite for p.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be testing problems for I_1 and I_2. $\mathcal{T}_1$ can be *finitely reduced* to $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \leq_F \mathcal{T}_2$, if there exist some computable functions $e : \mathcal{T}_1 \longrightarrow \mathcal{T}_2$ and $t : 2^{I_2} \longrightarrow 2^{I_1}$ such that, for all $p_1 \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_2$, if $\mathcal{I}$ is a finite complete test suite for $e(p_1)$ then $t(\mathcal{I})$ is a finite complete test suite for p_1. □

Theorem 3: *(Testing reduction Theorem)* We have the following properties:

(a) $\leq_F$ is a preorder.
(b) Let $\mathcal{T}_1 \leq_F \mathcal{T}_2$. If there exists a finite suite derivation for $\mathcal{T}_2$ then there exists a finite suite derivation for $\mathcal{T}_1$.
(c) Let $\mathcal{T}_1 \leq_F \mathcal{T}_2$. If $\mathcal{T}_2 \subseteq$ `Class I` then $\mathcal{T}_1 \subseteq$ `Class I`.

□

Proof. We will consider that $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are testing problems for the sets of inputs I_1, I_2, and I_3, respectively.

Let us prove (a). First, we consider the reflexivity of $\leq_F$. Let $e : \mathcal{T}_1 \longrightarrow \mathcal{T}_1$ and $t : 2^{I_1} \longrightarrow 2^{I_1}$ be *identity* functions. We trivially have that for all $(C, E) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_1$, if $\mathcal{I}$ is a finite complete test suite for $e(C, E) = (C, E)$ then $t(\mathcal{I}) = \mathcal{I}$ is a finite test suite for (C, E). Thus, $\mathcal{T}_1 \leq_F \mathcal{T}_1$.

We prove the transitivity of $\leq_F$. Let us assume $\mathcal{T}_1 \leq_F \mathcal{T}_2$ and $\mathcal{T}_2 \leq_F \mathcal{T}_3$. There exist functions $e : \mathcal{T}_1 \longrightarrow \mathcal{T}_2$ and $t : 2^{I_2} \longrightarrow 2^{I_1}$ such that if $\mathcal{I}$ is a finite complete test suite for $e(C_1, E_1)$ then $t(\mathcal{I})$ is a finite complete test suite for (C_1, E_1), for all $(C_1, E_1) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_2$. Besides, there also exist functions $e' : \mathcal{T}_2 \longrightarrow \mathcal{T}_3$ and $t' : 2^{I_3} \longrightarrow 2^{I_2}$ such that if $\mathcal{I}$ is a finite complete test suite for $e'(C_2, E_2)$ then $t'(\mathcal{I})$ is a finite complete test suite for (C_2, E_2), for all $(C_2, E_2) \in \mathcal{T}_2$ and $\mathcal{I} \subseteq I_3$. Let $e'' = e' \circ e$ and $t'' = t \circ t'$. Let $(C_1, E_1) \in \mathcal{T}_1$ and $\mathcal{I}$ be a finite complete test suite for $e''(C_1, E_1)$. This implies that $t'(\mathcal{I})$ is a complete test suite for $e(C_1, E_1)$. In turn, this implies that $t(t'(\mathcal{I}))$ is a complete test suite for (C_1, E_1). Thus, we can use functions e'' and t'' to make the required transformations between $\mathcal{T}_1$ and $\mathcal{T}_2$, and we have $\mathcal{T}_1 \leq_F \mathcal{T}_3$.

Next we consider (b). Since $\mathcal{T}_1 \leq_F \mathcal{T}_2$, there exist computable functions $e : \mathcal{T}_1 \longrightarrow \mathcal{T}_2$ and $t : 2^{I_2} \longrightarrow 2^{I_1}$ such that if $\mathcal{I}$ is a finite complete test suite for $e(C_1, E_1)$ then $t(\mathcal{I})$ is a finite complete test suite for (C_1, E_1), for all $(C_1, E_1) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_2$. Besides, let $d : \mathcal{T}_2 \longrightarrow 2^I$ be a finite suite derivation for $\mathcal{T}_2$. We know that d is computable, and $d(C_2, E_2)$ is a finite complete test suite for (C_2, E_2), for all $(C_2, E_2) \in \mathcal{T}_2$. Therefore, given $(C_1, E_1) \in \mathcal{T}_1$, $t(d(e(C_1, E_1)))$ is a finite complete test suite for (C_1, E_1). The composition of computable functions is computable, so $h = t \circ d \circ e$ is computable. Thus, h is a finite suite derivation for $\mathcal{T}_1$.

The property (c) is proved by using very similar arguments as in (b), though now we do not construct a finite suite derivation for $\mathcal{T}_1$. Instead, functions e and t are used to prove the existence of a finite complete test suite for each $(C_1, E_1) \in \mathcal{T}_1$ as follows. Since $\mathcal{T}_2 \subseteq$ `Class I`, there exists a finite complete test suite for $e(C_1, E_1) \in \mathcal{T}_2$. By applying function t, this suite can be transformed into a finite complete test suite for $(C_1, E_1) \in \mathcal{T}_1$. Thus, $\mathcal{T}_1 \subseteq$ `Class I`. □

The relation $\leq_F$ lets us relate testing problems with each other. In this sense, it reminds the reductions of computability and complexity theory, where a computation problem is transformed into another problem.[10] In our case, we check whether finding a finite complete test suite in a given scenario can be achieved by means of finding a finite complete test suite in another one.

We illustrate $\leq_F$ in Case Study 5.3 of the next section, where the problems of testing FSMs, EFSMs and TEFSMs are related by means of testing reductions. Due to some finiteness constraints imposed to EFSMs and TEFSMs, it turns out that each computation formalism can *simulate* the other two ones. However note that, in general, a testing reduction does *not* consist in mapping one computation formalism into another, but in mapping the *border* between correctness and incorrectness from one problem to another. This is illustrated later in the same case study with an additional short example, where testing Turing Machines is reduced to testing Finite Automata when faults follow a given form and no size constraint is assumed (so the latter cannot simulate the former). Section 5 also presents the more elaborated Case Study 5.4, where testing several kinds of machines involving an increasing magnitude (some denoted by *infinite* computation formalisms) are reduced into the problem of testing deterministic FSMs with no more that n states (which is a finite computation formalism). Moreover, in some cases transitions are labeled with magnitude values from a continuous domain, and yet they can be reduced to that problem. Also, a series of reductions proves the inclusion of the problem involving n magnitudes into `Class I`, and this is used to develop a method to test a car control device.

5 Case studies

In this section we present four elaborated examples where the notions proposed in the paper are applied to different scenarios. They include the discovery of some interesting properties for some well-known testing frameworks as well as other useful ones.

5.1 Case study: Hennessy's Framework

Let us study the testability of a classical testing framework in terms of the testability notions proposed here. In particular, we will study the relation of the semantic framework proposed by M. Hennessy in [17] with `Class II`. In that framework, a theory of concurrent

10. In fact, a testing problem also represents a kind of computability problem.

processes is presented. Systems are defined as processes, and each process implicitly defines an associated *labeled transition system* denoting how actions are iteratively performed. There is no distinction between inputs and outputs, we just consider *actions*. In order to check if a process *passes* a test, *all* possible computations of the process synchronized in parallel with the test (that is, all possible sequences of transitions which may be taken by the process in response to the test) are studied. There are two notions of passing a test: *must* and *may* test passing. A process passes a test in the *must* sense if all possible computations are considered correct by the test. A process is passed in the *may* sense if there exists at least one computation considered correct by the test. In both cases, the outcome of a test is either acceptance or failure.

Before presenting in more detail other relevant aspects of that framework, it is worth to mention that the effect of the *non-determinism* on whether we can consider a complete test suite as complete is different in that framework and ours. We illustrate it with an example defined in our setting. Let $f_1(i) = \{a\}$ and $f_2(i) = \{a, b\}$. That is, if input i is given, then f_1 will always reply a, whereas f_2 non-deterministically replies either a or b. In our framework we consider that $\{i\}$ is not a complete test suite for $(\{f_1, f_2\}, \{f_1\})$ because, after applying i to either f_1 or f_2, its reply does not *necessarily* let us decide whether it was f_1 or f_2 (the reply a does not allow us to determine which one the IUT is). On the contrary, according to other views of non-determinism in testing such as Hennessy's, $\{i\}$ distinguishes f_1 and f_2 indeed. In terms of our setting, this can be justified as follows. Let us assume that, if we apply input i *many* times to f_2, then we will eventually observe a and we will eventually observe b, that is, we assume a *fairness* hypothesis. If we assume that all non-deterministic reactions of the IUT to a given tester interaction will be observed at least once after performing that interaction a *high enough* number of times, then applying i a *high enough* number of times will finally reveal whether the IUT is f_1 or f_2: if b is eventually replied then it is f_2 else it is f_1. There are several ways to effectively model this particular non-determinism view into our general framework. Perhaps the simplest and more direct one consists just in representing this framework as a *deterministic* environment where, in particular, there exists a (single) output called "$\{a\}$" as well as a (single) output called "$\{a, b\}$". Also, we assume that our distinguishing relation $\not\eqsim$ distinguishes output $\{a\}$ from output $\{a, b\}$, because the repetition of any interaction producing one of them will *eventually* distinguish it from the other, so in general we consider $\not\eqsim$ iff $\neq$. Thus, the (deterministic) functions $f_1(i) = \{\{a\}\}$ and $f_2(i) = \{\{a, b\}\}$ are distinguished by input i in our framework, so $\{i\}$ is a complete test suite.

Let us introduce other important aspects of the framework given in [17] which must be considered here. The author provides a testing semantics for a process algebra at three levels: an operational semantics, a denotational semantics, and an axiomatic semantics. It is proved that the three semantics are equivalent. The author considers a set of actions A, and an algebra with the typical operators of a process algebra. Then, an operational semantics is defined as well as a testing semantics. One of the key points is that the testing semantics can be characterized just by using the operational semantics. In this way, the author defines the *acceptance* of a process p after a trace $s \in A^*$ as $\mathcal{A}(p, s) \subseteq \mathcal{P}(\mathcal{P}(A))$, where $\mathcal{P}(X)$ denotes the powerset of X. This set characterizes the non-deterministic possibilities of a process p after executing the trace s. For instance, $\mathcal{A}(p, \epsilon) = \{\{a\}, \{b\}\}$ indicates that the process can initially execute internal actions, and then it reaches either a state where only a is possible, or a state where only b is possible. On the other hand, $\mathcal{A}(q, \epsilon) = \{\{a, b\}\}$ indicates that, after possibly executing internal actions, the process q is ready to offer a *and* b, so the environment (e.g. a test) can choose both actions. The process p can be written syntactically as $a \oplus b$ (internal choice), whereas q can be written as $a + b$ (external choice). In terms of tests, we say that the test $T = a; \checkmark$ is passed by process q but not by p (p fails the test T) under the *must testing semantics*. As mentioned before, a test is passed under these semantics if all possible computations of the process with the test are passed. It is clear that a single execution of the test T and p might not fail, because the execution of a is possible in p. However, if T is repeated, then the process p will eventually choose the other branch, and then the test T will fail.

Next we show how the Hennessy's framework can be expressed in our framework and related to `Class II`. Here we will assume that A is a finite set (this restriction is only necessary to prove our result regarding `Class II`). We will consider the denotational view of processes and the must testing semantics. Under these assumptions, a process is just a tree where nodes are decorated with acceptance sets $\mathcal{A} \subseteq \mathcal{P}(A)$ that have to be closed under union and convexity, and arcs are labeled by some action $a \in A$. Moreover, if a node is decorated with set $\mathcal{A}$, then it has a unique outgoing arc labeled with each $a \in \cup_{A \in \mathcal{A}} A$. Under this view, a tree t can be seen as a function from the set of traces A^* to $\mathcal{P}(\mathcal{P}(A))$ where $t(s)$ is the decoration of the node reached after processing the sequence s. We assume that function t also fulfills the following restriction: for any $a \in A$ and $s \in A^*$, $t(sa) = \varnothing$ iff $a \notin \cup_{A \in \mathcal{A}} A$, where $\mathcal{A} = t(s)$. The set of all of these trees is denoted by $\mathbf{fAT_S}$.[11]

In order to represent Hennessy's model into our setting, we consider that our set of inputs I is the set of traces A^*, and the outputs set is the set $O = \{\mathcal{A} \mid \mathcal{A} \subseteq$

11. We have chosen to represent the set $\mathbf{fAT_S}$ for clarity reasons in this example, as there is only one kind of nodes in these trees. However, it would not be very hard to consider the sets $\mathbf{AT_S}$ or $\mathbf{AT}$. In these cases, the definitions of the functions would be a bit more complicated, because there are two kinds of nodes (open and closed), and there are some restrictions that apply to open nodes.

$\mathcal{P}(A),\ \mathcal{A}$ is closed under union and convexity$\}$

The distinguishing relation $\not\equiv$ is the trivial one. We define our computation formalisms set as $C = \{f_t \mid t \in \mathbf{fAT_S}\}$, where $f_t(s) = \{\mathcal{A}\}$ iff $t(s) = \mathcal{A}$ (recall that we consider $\mathcal{A}$ as a *single* output of f_t in our setting). The set E consists in some function $f_0 \in C$ denoting some tree $t_0 \in \mathbf{fAT_S}$, so $E = \{f_0\}$.

Next we prove $(C, E) \in$ `Class II`. Initially we will consider the set $CF = \{f_t \mid t \in \mathbf{fAT_S}, t \text{ finite}\}$,[12] and we will prove $(CF \cup \{f_0\}, \{f_0\}) \in$ `Class II`. The more general case where f_0 is also compared with functions denoting infinite trees will be tackled afterwards. For any $k \in \mathbb{N}$, we consider the following equivalence relation: $t \equiv_k t'$ iff $t(s) = t'(s)$ for all $s \in A^{k'}$ with $k' \leq k$.[13] It is easy to check that $\equiv_k$ is indeed an equivalence relation. For any $t \in CF$, the equivalence class induced by t for $\equiv_k$ is denoted by $[t]_k$, and the set of all equivalence classes for $\equiv_k$ will be denoted by CF^k.

Let us introduce the sets $C_0, C_1, \ldots$ required in the definition of `Class II`. The set C_0 consists of one representative of each equivalence class of $\equiv_0$, though we force to choose f_0 as representative of its equivalence class. Since the alphabet is finite, the number of equivalence classes induced by $\equiv_k$ is finite for all $k \in \mathbb{N}$. Let $t_1, \ldots, t_g$ be arbitrary elements of each equivalence class $c \in CF^0$ with $c \not\equiv [t_0]_0$. Then, $C_0 = \{f_0, f_{t_1}, \ldots, f_{t_g}\}$.

Similarly, C_{i+1} is built by considering one representative of all equivalence classes of $\equiv_{i+1}$, but forcing that the representatives already chosen in C_i are also the representatives of their corresponding equivalence classes of CF^{i+1} in the set C_{i+1}. So for any $c \in CF^{i+1}$:

- If there is some $f_t \in C_i$ such that $t \in c$, then we pick f_t as representative of c in C_{i+1}, that is, $f_t \in C_{i+1}$.
- Otherwise, we choose any $t \in c$ and make $f_t \in C_{i+1}$.

It is clear that $C_i \subseteq C_{i+1}$, $CF = \bigcup_{i \in \mathbb{N}} C_i$, and the set of inputs A^i is a complete test suite for (C_i, E). Since A is finite, the set A^i is a finite complete test suite.

Let $s_i = |C_i|$. It is clear that $\lim_{n\to\infty} s_n = \infty$. Let $n, l \in \mathbb{N}$ be such that $l > n$. The set of inputs A^n can only distinguish elements that are in different classes of $\mathbf{fAT_S}^n$. Let us consider the set $T = \{f \mid f \in C_l, f \in [t_0]_n\}$. Then, $|\{(f_0, f') \mid \mathtt{di}(f_0, f', A^n),\ f' \in C_l \backslash \{f_0\}\}| = s_l - |T|$.

Next we identify an upper bound of $|T|$. For any $k \geq n$, let us consider the set $T^k_{t_0} \subseteq C_k$ containing all processes that, at least, include all traces of t_0 whose length is n:

$$T^{k,n}_{t_0} = \{f \mid f \in C_k, \forall s \in A^n : f_0(s) \neq \varnothing \Rightarrow f(s) \neq \varnothing\}$$

Let us prove $|T^{l,n}_{t_0}| = |T^{n,n}_{t_0}||T|$. In order to do so, we define a bijection between $T \times T^{n,n}_{t_0}$ and $T^{l,n}_{t_0}$. We define $\mathtt{b} : T \times T^{n,n}_{t_0} \mapsto T^{l,n}_{t_0}$ as follows. Let $(f_1, f_2) \in T \times T^{n,n}_{t_0}$, and let us consider the following function:

$$f(s) = f_2(s) \text{ if } |s| \leq n \qquad f(s) = f_1(s) \text{ if } |s| > n$$

It is easy to check that $f \in T^{l,n}_{t_0}$. We can define $\mathtt{b}(f_1, f_2)$ as the unique representative of f in C_l. It is not difficult to prove that $\mathtt{b}$ is indeed a bijection. First, we prove that $\mathtt{b}$ is surjective. Let us consider $f \in T^{l,n}_{t_0}$. Let us build the function f' such that $f'(s) = f_0(s)$ for any trace s such that $|s| \leq n$ and $f'(s) = f(s)$ for any trace s such that $|s| > n$. Then let us consider f_1 as the unique representative in C_l of $[f']_l$. It is clear that $f_1 \in T$. Now let us consider f_2 as the unique representative in C_n of f. Since $f \in T^{l,n}_{t_0}$, we have $f_2 \in T^{n,n}_{t_0}$. Then, by definition $\mathtt{b}(f_1, f_2) = f$. Second, let us prove that $\mathtt{b}$ is injective. Let us consider $(f_1, f_2), (f'_1, f'_2) \in T \times T^{l,n}_{t_0}$ such that $\mathtt{b}(f_1, f_2) = \mathtt{b}(f'_1, f'_2)$. Since $f_1, f_1 \in T$, we have $f_1(s) = f_2(s) = f_0(s)$ for any trace s such that $|s| \leq n$. By the definition of $\mathtt{b}$, we obtain $f_1(s) = f'_1(s)$ for any trace $n < |s| \leq l$; so $[f_1]_l = [f'_1]_l$. Since $T \subseteq C_l$ and there is only one representative per class in C_l, we obtain $f_1 = f'_1$. By the definition of $\mathtt{b}$ we obtain $f_2(s) = f_2(s)$ for any trace s such $|s| \leq n$. So $[f_2]_n = [f'_2]_n$. Since $f_2, f'_2 \in T^{n,n}_{t_0} \subseteq C_n$ and there is only one representative per class in C_n, we obtain $f_2 = f'_2$.

Then we obtain $|T| = \frac{|T^{l,n}_{t_0}|}{|T^{n,n}_{t_0}|}$. Since $T^{l,n}_{t_0} \subseteq \mathbf{fAT_S}^l$, we infer $|T| \leq \frac{s_l}{|T^{n,n}_{t_0}|}$. It is also clear that $\lim_{n\to\infty} |T^{n,n}_{t_0}| = \infty$. So

$$\mathtt{d\text{-}rate}\left(A^k, C_l, E\right) = \frac{|\{(f_0, f') | \mathtt{di}\left(f_0, f', A^k\right), f' \in C_l \backslash \{f_0\}\}}{s_l - 1} = \frac{s_l - |T|}{s_l - 1} \geq$$

$$\geq \frac{s_l - \frac{s_l}{|T^{n,n}_{t_0}|}}{s_l - 1} = \frac{s_l\left(1 - \frac{1}{|T^{n,n}_{t_0}|}\right)}{s_l - 1}$$

Let $\epsilon < 1$. Let us consider the expression $S_k = 1 - \frac{1}{|T^{k,k}_{t_0}|}$. The expression $\frac{s_n S_k}{s_n - 1}$ is a decreasing succession whose limit is S_k when n tends to infinity. So, in order to make $\frac{s_n S_k}{s_n - 1} > \epsilon$, it is enough to find a value of S_k such that $S_k > \epsilon$. Since the succession S_k converges to 1, there exists n_0 such that $S_{n_0} \geq \epsilon$. Hence, for all $l \geq n_0$ we obtain $\frac{s_l S_{n_0}}{s_l - 1} > \epsilon$. We deduce $(CF, E) \in$ `Class II`.

In order to prove that $(C, E) \in$ `Class II`, let us consider $CI = C \backslash CF$. The set CI is countable, so let us consider a numeration $c_0, c_1, c_2, \ldots$ of CI. Now let us consider the sequence $C'_i = C_i \cup \{c_0, \ldots, c_i\}$. In each C'_i there are $i + 1$ more elements that in C_i. At worst, those elements are not distinguishable from f_0. So

$$\mathtt{d\text{-}rate}\left(A^k, C_l, E\right) \geq \frac{s_l - |T| - (l+1)}{s_l - 1} \geq$$

$$\geq \frac{s_l - \frac{s_l}{|T^{n,n}_{t_0}|} - (l+1)}{s_l - 1} \geq \frac{s_l(1 - \frac{1}{|T^{n,n}_{t_0}|})}{s_l - 1} - \frac{l+1}{s_l - 1}$$

In this case, we also have that $\frac{l+1}{s_l - 1}$ is a succession whose limit is 0. Hence we infer $(C, E) \in$ `Class II`.

5.2 Case study: Timeout Machines

Our second case study also tackles an elaborated (and somehow surprising) case of inclusion in `Class II`. Let

12. A tree is finite if its number of traces is finite, formally $|\{s \mid t(s) \neq \varnothing\}| < \infty$.

13. A^k is the set of all traces whose length is no longer than k.

us consider *FSMs with rational timeouts* (in short, *timeout machines* hereafter). These machines work as follows. We consider that all states have exactly one outgoing transition labeled with each input and some output (thus they are fully-specified and deterministic). In addition, all states also have one *timeout transition*, which is triggered when the time spent in that state reaches a given rational value r with $0 < r \leq M$ for some given rational M. The transition leads the machine to another state. Timeout machines are stimulated by means of sequences where some waiting times strictly alternate with actions where some input button is pressed and some output is replied (the sets of input and output symbols are finite). Note that, contrarily to our previous examples, now the choices of the tester *at each step* of the stimulation of the IUT are infinite: the tester can press a button (finite set of choices) or wait for any rational time within an interval (infinite set of choices). This makes testing particularly difficult: at each step of the execution of the IUT, any finite number of tests can check only what happens in a finite number of situations, but there are infinite choices we could take at this step. Thus, any finite testing plan checks a 0 *proportion* of all cases that must be checked at each point (a finite amount out an infinite amount which must be checked). Yet this case is in `Class II`, as we prove next.[14]

Let $E = \{f\}$, where f represents the behavior of a timeout machine (actually many, as it represents all machines with the same behavior). Let m_f be the *minimum* timeout machine, i.e. the one having less states, behaving as f (note that it is unique up to isomorphism), and let it have n states. Without loss of generality, let us suppose that all timeouts of m_f are multiple of some $\frac{1}{v}$. Note that, even if all rational timeouts in m_f have different denominators after simplified, all of them can be expressed as a multiple of some $\frac{1}{v}$ where v is the multiplication of all the denominators of timeouts of m_f.

Let C consist of all functions denoting timeout machines. Let us remark that machines represented by functions in C can have any number of states. The set of inputs which can be received by each function $f \in C$, denoted as usual by I, is the set of sequences which strictly alternate input symbols and rational delays as mentioned before.

Let us define the required sets $C_1 \subseteq C_2 \subseteq ...$ such that $\bigcup_{i \in \mathbb{N}^+} C_i = C$. We define the functions included in each set C_i as follows. Let the set A_i^j consist of all deterministic partially-specified timeout machines with the form of *trees* where the root denotes the initial state, all branches have length i, all non-leaf nodes denote fully specified states (i.e. they have one timeout transition as well as one transition labeled with each input and some output), all leaf nodes denote unspecified states (i.e. they have no outgoing transition), and all timeout transitions are labeled with rational numbers $0 < r \leq M$ which are multiple of $\frac{1}{j}$. Hereafter this kind of machines will be denoted as *partially-specified tree-form timeout machines*. We will say that a function f' is *consistent* with some partially-specified tree-form timeout machine m' of depth k if the following condition holds: f' behaves as some minimum timeout machine m'' where all sequences of k consecutive transitions from the initial state are labeled with the same inputs, outputs, and timeouts as some sequence of transitions available in m' from its initial state.

Similarly as in the proof of $(C_2, E_2) \in$ `Class II` in Example 4, let us consider that the *minimum function* fulfilling a given criterion H is the function fulfilling H which can be represented with a machine (in this case, a timeout machine) with the minimum number of states (we assume that ties between timeout machines with the same size are solved in any arbitrary fix way). For each $m \in A_i^j$, let $f_m \in C$ be the *minimum* function such that (a) it is consistent with m and (b) *all* timeouts of the minimum timeout machine represented by f_m are multiple of $\frac{1}{j}$ (not only those included by m). Then we define $C_i = \{f\} \cup \{f_m \mid m \in A_i^{i!}\}$ (recall that f is the specification function). Let us check that $\bigcup_{i \in \mathbb{N}^+} C_i = C$ and $C_i \subseteq C_{i+1}$ for $i \in \mathbb{N}^+$.

First we see that $C_i \subseteq C_{i+1}$ for all $i \in \mathbb{N}^+$. All functions of C_i but f are constructed from the partial tree-form machines of $A_i^{i!}$. On the one hand, note that all timeout values used in functions in C_i are allowed by C_{i+1} (we have $\frac{1}{i!} = \frac{1}{(i+1)!} \cdot (i+1)$, so all timeouts used in machines represented by functions of C_i can also be expressed as multiples of $\frac{1}{(i+1)!}$). On the other hand, note that the minimum function which is consistent with some partially-specified tree-form machine used to build C_i will also be so for some partially-specified tree-form machine used to build C_{i+1}. In particular, since the set of tree-form machines considered in the construction of functions in C_{i+1} covers *all* possible ways to interact during a $(i+1)$-long interaction (where timeouts are multiple of $\frac{1}{(i+1)!}$), one of them must coincide with the behavior of each function which was the minimum one being consistent with some tree-form machine considered in the construction of functions in C_i (where timeouts are multiple of $\frac{1}{i!} = \frac{1}{(i+1)!} \cdot (i+1)$). Thus $C_i \subseteq C_{i+1}$.

Now we show $\bigcup_{i \in \mathbb{N}^+} C_i = C$. Note that, as we commented before when we introduced the specification function f, forcing all timeouts of *each* machine to be multiple of the *same* rational number $\frac{1}{i}$ for some $i \in \mathbb{N}^+$ does not prevent us from denoting any function in

14. One could consider that, since measure devices are digital and only discrete magnitudes can be considered in practice, the difficulty of dealing with dense time (rational times) does not appear in practice. However, in an environment where delays can take hours or nanoseconds, considering that the set of possible times is discrete (and hence, also finite if closed intervals are considered) does not help practical testing because the number of times to be checked is astronomical anyway (this is similar to assuming that computers do not compute recursive languages but *regular languages* because, since the memory of a real computer is limited, a computer can be only in a finite set of states, so it can be fully denoted by some finite automaton. This is technically true, but this view does not help to reason about programs or solve any problem). On the contrary, if we develop a testing methodology under the assumption that the time is dense, then it will *necessarily* assume that we cannot check *all* times in practice.

C. Moreover, all timeouts multiple of some $\frac{1}{i}$ are also multiple of $\frac{1}{i!}$, $\frac{1}{(i+1)!}$, ..., so all sets C_i, C_{i+1}, ... let represent functions where all timeouts are multiple of some $\frac{1}{i}$. Let us prove that any function $f' \in C$ is included in some C_i, either because it is the minimum machine being consistent with some of the partially-specified tree-form machines considered in the construction of C_i, *or* because it is f (the specification). Note that f is explicitly added to all C_i, so this case is trivial. Let us consider $f' \neq f$. Without loss of generality, let us assume that f' represents some timeout machine where all timeouts are multiple of some $\frac{1}{d!}$ with $d \in \mathbb{N}^+$. Note that f' is consistent with a *single* partially specified machine of $A_i^{i!}$ if $i \geq d$, because *all* possible trees of transitions of depth i where timeouts are multiple of $\frac{1}{i!}$ are represented in $A_i^{i!}$, and exactly one of them is consistent with f'. Thus, if a given function $f' \in C$ is not in some C_i with $i \geq d$, it is because f' is not the *minimum* function being consistent with the single partial machine of $A_i^{i!}$ it is consistent with. Could f' be such that, for *all* $i \geq d$, it is not the minimum function consistent with the single partial machine of $A_i^{i!}$ it is consistent with? Let us suppose that the minimum timeout machine behaving as f' has J states. Let K be the number of minimum timeout machines such that all of their timeouts are multiple of $\frac{1}{d!}$ and have J or less states (the set of them is finite because all timeouts must be at most M). Note that, by iteratively considering the functions added by each $C_1, C_2, C_3, \ldots$, the number of times the minimum function being consistent with some partially-specified tree-form machine (where all timeouts are multiple of $\frac{1}{d!}$) is selected *but* it is not f', is at most K. Thus, f' is eventually added to some C_i.

Let $\mathcal{I}_d \subseteq I$ be a test suite consisting of all stimulation sequences of the form $a_1 \ldots a_d$ where, for all $1 \leq k \leq d$, a_k is either an input symbol of the machine or a time delay not greater than M and multiple of $\frac{1}{d}$. Note that any subsequence of $a_1 \ldots a_d$ consisting only of consecutive time delays can be trivially converted into a single delay amounting all of them together. Similarly, any sequence of consecutive input symbols can be translated into a sequence where a 0 delay is put between each pair of input symbols. Thus, any sequence in $\mathcal{I}_d$ can be expressed as a sequence where input symbols and time delays strictly alternate, as required for $\mathcal{I}_d \subseteq I$. Let us find a lower bound of the distinguishing rate of $\mathcal{I}_d$ for each pair $(C_i, \{f\})$.

By the construction of sets $C_1, C_2, \ldots$, we can see the following property. Let us consider any combination of *outputs* (one for each input/output transition with its corresponding input) and *timeouts* (one for each timeout transition) which could label the first $d \leq i$ transitions reachable in a timeout machine represented by a function of C_i. The number of functions of C_i representing a machine with *that* combination of timeouts and outputs along the first d steps is the *same* as the number of functions of C_i representing any *other* combination of timeouts and outputs during the first d steps. Thus, the *proportion* of functions of C_i having some given combinations of outputs and timeouts in all transitions traversed during the first d steps is equal to the proportion of machines of the set $A_d^{i!}$ (i.e. the set of partial tree-form machines of depth d where all timeouts are multiple of $\frac{1}{i!}$) having the same combinations. Since the capability of $\mathcal{I}_d$ to detect the incorrectness of a function depends only on the combination of outputs and timeouts labeling the transitions of the machine represented by this function during the first d steps, we conclude that the proportion of functions from C_i which are (un-)distinguished by $\mathcal{I}_d$ matches exactly the proportion of partial tree-form machines of $A_d^{i!}$ which are (un-)distinguished by $\mathcal{I}_d$.

We could think that $\mathcal{I}_d$ can distinguish f from any function $f' \in C_d$ where the combination of outputs and timeouts of the machine represented by f' along the first d reachable transitions is not *exactly* the combination of f. If this were true, then $\mathcal{I}_d$ would left undistinguished only those functions which have exactly the *same* timeouts and produce the same outputs for each input during their first d steps. Thus, the distinguishing rate of $\mathcal{I}_d$ for $(C_i, \{f\})$ with $i \geq d$ would be at least one minus the proportion of functions in C_i having the same timeouts and outputs as f for their first d steps (which, by the property introduced in the previous paragraph, would be one minus the proportion of partial tree-form machines in $A_d^{i!}$ having the same timeouts and outputs as f for their first d steps). Unfortunately, there are some functions $f' \in C_i$ representing machines that do not have the same combination of outputs and timeouts during the first d steps as f, and yet $\mathcal{I}_d$ might not be able to identify their incorrectness. In particular, when the test suite $\mathcal{I}_d$ is applied, some alternative assignments of outputs and timeouts to the first d transitions might produce a response being undistinguishable from that of f.

On the one hand, let us suppose that two states produce exactly the same output for each input, and one of them leads to the other through a timeout transition. Let us suppose that both states are consecutively traversed by some execution of d steps. Note that $\mathcal{I}_d$ might not be able to discover when the timeout from one state to the other is taken. In particular, this is the case if the remaining steps up to the d-th step do not produce a different behavior from both states (which obviously does not imply than longer interactions could not distinguish them indeed). A necessary (not sufficient) condition for $\mathcal{I}_d$ not to be able to distinguish the behavior of both states, and thus not to discover when the timeout is taken, is that two states produce the same outputs for all inputs and the timeout of one of them leads to the other one.

On the other hand, let us note that the time delays produced in $\mathcal{I}_d$ could be too sparse and skip some relevant time frames where incorrect behaviors might happen, so they could remain unobserved by $\mathcal{I}_d$. Two kinds of these problems may occur:

(a) The IUT could move to another state where the behavior is wrong (i.e. the output replied for some input

is different to the reply defined in the specification after the same previous interaction), and stay in that wrong state for a *very short* time, so the sparse times checked by $\mathcal{I}_d$ skip the time frame where the IUT is at this state and do not discover it. A necessary condition for this is that the IUT autonomously leaves this wrong state before $\frac{1}{d}$ units of time: otherwise, some test case in $\mathcal{I}_d$ would produce some input during the stay in that state, and its wrong behavior would be eventually discovered. Thus, the timeout of the wrong state must be less than $\frac{1}{d}$.

(b) The sparse times checked by $\mathcal{I}_d$ might not be able to detect the *precise* time when the IUT takes some timeout, even when the responses of both states differ for at least one input. In fact, since the time delays produced in $\mathcal{I}_d$ between consecutive inputs are at least $\frac{1}{d}$, they just let us to detect that each IUT timeout was taken within some time frame of width $\frac{1}{d}$. Hence the actual timeout could be taken at *any* point within that time frame, although only the specific time defined by the specification for the corresponding state is valid. An upper bound of this uncertainty zone, where we might not know where the actual timeout is, is the area between $-\frac{1}{d}$ and $\frac{1}{d}$ time units from the actual timeout required by the specification at that state. Note that the *correct timeout* is included in this area.

If the test suite $\mathcal{I}_d$ is used to test $(C_i, \{f\})$ then, how many possible timeouts of machines represented by functions of C_i could fit within the uncertainty areas mentioned in (a) or (b) (which amount a size of $\frac{1}{d}$ and $\frac{2}{d}$, respectively)? Taking into account that C_i contains machines where all timeouts are multiples of $\frac{1}{i!}$, an upper bound of the number of possible timeout values within the $\frac{1}{d}$ critical area (respectively, the $\frac{2}{d}$ critical area) is equal to $\frac{1/d}{1/i!}$ (resp. $\frac{2/d}{1/i!}$). On the other hand, since all timeouts t must fulfill the condition $0 < t \leq M$, the number of all possible timeouts in each state of a machine of C_i is $\frac{M}{1/i!}$. By dividing the two previous expressions, we infer that an upper bound of the *proportion* of available timeouts which fall into the critical area mentioned in (a) is $\frac{1/d}{M} = \frac{1}{dM}$, and the proportion in the critical area mentioned in (b) is $\frac{2/d}{M} = \frac{2}{dM}$.

Let us suppose that some IUT timeout is deviated from the timeout actually required by the specification for the corresponding state. If the deviated IUT timeout is *out* of the critical areas considered in (a) and (b), then it will produce a behavior which will be able to be detected as incorrect by $\mathcal{I}_d$ provided that the state reached after the timeout replies a different output for at least one input. On the one hand, if the timeout mentioned in (a) is higher than $\frac{1}{d}$ then at least one test case of $\mathcal{I}_d$ will observe the IUT behavior at the new state. Thus, if the reply of that state to some input is different from the reply of the previous state, it will be detected. On the other hand, if the timeout mentioned in (b) is further than $\frac{1}{d}$ units from the timeout required by the specification, then this excessive deviation will be discovered by at least one test case of $\mathcal{I}_d$, provided that the reply before and after the timeout is taken is different for some input. We conclude that, if there is a wrong timeout but the reply of the machine before and after the timeout is different for at least one input, then this fault can remain undetected by $\mathcal{I}_d$ only if the timeout is *within* the critical areas mentioned in (a) or (b), which account a proportion of $\frac{1}{dM}$ and $\frac{2}{dM}$ respectively.

All in all, we conclude that a necessary (but not sufficient) condition for $\mathcal{I}_d$ not being able to detect a fault in an incorrect function $f' \in C_i$ is that, for all state s of the timeout machine represented by f' such that s is reachable by some sequence in $\mathcal{I}_d$, either (i) its timeout is within the critical areas (a) or (b) (amounting a total proportion of $\frac{3}{dM}$); or (ii) the state s' reached by the timeout transition leaving s produces, for all inputs, the same outputs as s. Note that, if for some state s reachable by $\mathcal{I}_d$ neither (i) nor (ii) hold, then the timeout of s is produced at a time such that $\mathcal{I}_d$ observes what happens before and after it for at least one point at each side (due to (i) not holding), and the behavior of both consecutive states is different for at least one input (due to (ii) not holding), so $\mathcal{I}_d$ discovers that a timeout is taken at an unexpected time and hence a fault is detected. It is worth to mention that, if the timeout machine represented by f' is such that all transitions reachable in d steps are labeled with the *same* outputs and timeouts as in the timeout represented by f, then f' also fulfills the condition stated at the beginning of this paragraph, because the correct timeout value is also within the critical area considered in (b) (note that such f' could be incorrect or not). We conclude that a function $f' \in C_i$ can be incorrect and not be detected as incorrect only if (i) and (ii) hold for all the states reachable by $\mathcal{I}_d$ in the timeout machine represented by f'.

Thus, a lower bound of the distinguishing rate of $\mathcal{I}_d$ for $(C_i, \{f\})$ is given by the proportion of functions of C_i where at least one state reached by $\mathcal{I}_d$ is such that either its timeout is *out* of these critical areas, or the response of that state for some input is *different* from the response of the state reached after the timeout. In order to calculate this, let us count the proportion of functions of C_i where all timeouts taken during the execution of all sequences of $\mathcal{I}_d$ are *within* the critical areas *or* lead to a state where all inputs are replied with the same outputs as in the previous state. Let N be the number of input symbols and L be the number of output symbols (we assume $N, L \geq 2$). The proportion of functions in C_i where a given state s produces, for all inputs, the same outputs as the state it reaches through its timeout transition, is $\frac{1}{L^N}$. We deduce that the proportion of functions where, for a given state s reached by $\mathcal{I}_d$, either its timeout is within the critical areas mentioned before *or* for all inputs it produces the same outputs as the state reached by its timeout transition is $\frac{3}{dM} + \frac{1}{L^N} - \frac{3}{dM} \cdot \frac{1}{L^N}$.

Let us consider $i \geq d$. In order to compute the *proportion* of functions of C_i which fulfill some property concerning *only* their first d steps, we just have to calculate the proportion of trees of length d which fulfill the

considered property. As mentioned earlier, the reason is that the number of functions in C_i being exactly as each of these d-depth trees during their first d steps is the same for all of these trees, so a proportion calculated for the set of all partially-specified tree-form machines of depth d steps is the same for the set of all partially-specified tree-form machines of depth $i \geq d$.

Taking all the previous considerations into account, we can compute a lower bound of the distinguishing rate of $\mathcal{I}_d$ for C_i with $i \geq d$ as follows. We will calculate an upper bound of the proportion of incorrect functions from C_i which might not produce any incorrect response when all sequences in $\mathcal{I}_d$ are given. Let $v(k)$ denote the proportion of partial tree-form machines of depth k where, for all state s of the tree, either

- there is some previous input/output transition along its branch which departs from a state whose timeout is less that $\frac{1}{d}$ (note that this input/output transition might not be taken by $\mathcal{I}_d$ because all delays between inputs are at least $\frac{1}{d}$, so $\mathcal{I}_d$ could never reach s), or
- if not (so s can be reached indeed) then either its timeout is within the critical $\frac{3}{dM}$ area or its responses for all inputs are exactly the same as in the previous state of the branch.

Note that, if $i \geq d$, then $v(d)$ provides an upper bound of the proportion of incorrect functions in C_i which cannot be distinguished from f when all test cases in $\mathcal{I}_d$ are applied, so we have $\texttt{d-rate}\,(\mathcal{I}_d, C_i, \{f\}) \geq 1 - v(d)$.

Let us recursively define $v(k)$. Let $v(1) = \frac{3}{dM} + \frac{1}{L^N} - \frac{3}{dM} \cdot \frac{1}{L^N}$. Regarding the recursive case, let

$$v(k+1) = \tfrac{1}{dM} \cdot v(k) + \tfrac{dM-1}{dM} \cdot \left(\tfrac{2/dM}{(dM-1)/dM} + \tfrac{1}{L^N} - \tfrac{2/dM}{(dM-1)/dM} \cdot \tfrac{1}{L^N} \right) \cdot v(k)^{N+1}$$

Let us explain the previous expression, which denotes an upper bound of the proportion of trees of depth $k+1$ which are incorrect but could not be detected as such by $\mathcal{I}_d$. Let s be the state at the root of the tree.

(a) The first line of the expression denotes the proportion of these trees where, in particular, the timeout of s is less than $\frac{1}{d}$. In this case, the proportion of trees where this happens ($\frac{1}{dM}$) is multiplied by the proportion of all cases where the subtree reached from s through its timeout also fulfills the property ($v(k)$). Note that the rest of subtrees reached from s in a single transition are reachable through *input/output* transitions, but these transitions could not be taken by $\mathcal{I}_d$ in the *worst case* (as the timeout of s is too small for $\mathcal{I}_d$), so these subtrees are not required to fulfill the property (recall that we are computing an *upper bound* of the proportion of incorrect but undetected functions). On the contrary, the timeout transition of s certainly can be taken, so we must include the $v(k)$ term indeed.

(b) The second line denotes the proportion of trees where, in particular, the timeout of s is higher than $\frac{1}{d}$. In this case, the proportion of trees like this ($\frac{dM-1}{dM}$) is multiplied by the proportion of all cases where some source of potentially undetected faults happens at s. This proportion is the addition of the proportion of timeouts which are at most $\frac{1}{d}$ further from the required timeout at this point, conditioned to the fact that the timeout is not less than $\frac{1}{d}$ $\left(\frac{2/dM}{(dM-1)/dM}\right)$, plus the proportion of trees where s produces, for all inputs, the same outputs as the state reached by the timeout transition of s ($\frac{1}{L^N}$), minus the proportion of trees fulfilling both conditions (to avoid counting twice). Note that all subtrees reachable from s through a transition (either input/output or timeout) are required to fulfill the property in this case, because now $\mathcal{I}_d$ certainly can reach all of them (as the timeout is higher than $\frac{1}{d}$). Thus the previous amount is multiplied by the proportion of cases where *all* of these subtrees fulfill the property ($v(k)^{N+1}$).

Let $p = \frac{1}{dM}$ and $q = \frac{3}{dM} + \frac{1}{m}$. Note that we have $0 \leq v(k) \leq 1$ for all $k \geq 1$ because $v(k)$ is a *proportion*. Thus we have

$$\begin{aligned}
v(k+1) &= p \cdot v(k) + \\
&\quad (1-p) \cdot \left(\tfrac{2p}{1-p} + \tfrac{1}{L^N} - \tfrac{2p}{1-p} \cdot \tfrac{1}{L^N}\right) \cdot v(k)^{N+1} \leq \\
&\quad p \cdot v(k) + (1-p) \cdot \left(\tfrac{2p}{1-p} + \tfrac{1}{L^N}\right) \cdot v(k)^{N+1} = \\
&\quad p \cdot v(k) + \left(2p + \tfrac{1-p}{L^N}\right) \cdot v(k)^{N+1} \leq \\
&\quad p \cdot v(k) + \left(2p + \tfrac{1}{L^N}\right) \cdot v(k)^{N+1} \leq \\
&\quad q \cdot v(k) + q \cdot v(k)^{N+1} \leq \\
&\quad q \cdot v(k) + q \cdot v(k) = 2q \cdot v(k)
\end{aligned}$$

Let us define $v'(1) = 2q$ and $v'(k+1) = 2q \cdot v'(k)$. It is clear that $v'(k) \geq v(k)$ for all $k \geq 1$. Moreover, by induction over k, it is trivial to prove that $v'(k) = 2^k q^k$. Hence,

$$\begin{aligned}
&\texttt{d-rate}\,(\mathcal{I}_d, C_i, \{f\}) \geq 1 - v(d) \geq 1 - v'(d) = \\
&\quad 1 - 2^d q^d = 1 - 2^d \left(\tfrac{3}{dM} + \tfrac{1}{L^N}\right)^d
\end{aligned}$$

This lower bound of $\texttt{d-rate}\,(\mathcal{I}_d, C_i, \{f\})$ is strictly increasing with d, and it tends to 1 as d tends to infinity (note that $2^d \left(\frac{3}{dM} + \frac{1}{L^N}\right)^d$ tends to 0 as d increases because $L^N \geq 4$). Moreover, let us remark that this lower bound is valid for all $i \geq d$, so it does not depend on i provided that $i \geq d$. Thus, for any $\epsilon < 1$ there exists some d such that $\texttt{d-rate}\,(\mathcal{I}_d, C_i, \{f\}) > \epsilon$ for all $i \geq d$. We conclude $(C, E) \in \texttt{Class II}$.

5.3 Case study: Finite State Machines, Turing Machines and Finite Automata

We illustrate $\leq_F$ with examples. The first examples, dealing with FSM variants, will lie in manipulating *finite* computation formalisms and *mapping* computation formalisms into each other. Next, a more interesting example where computation formalism are infinite, dealing with Turing machines and finite automata, will be given to illustrate a case where mapping computation formalisms to each other is impossible, but there still exists a testing reduction because we can just map the *border* between correctness and incorrectness.

Let us compare FSMs, *extended finite state machines* (EFSMs) and *temporal EFSMs* (TEFSMs) in terms of testing.

EFSMs are FSMs where the behavior at each state is conditioned by the current values of some variables; the set of variables is finite. Each transition is enabled/disabled by a boolean condition over current variable values, and values are updated after taking each transition. We consider that the set of possible values for each variable is finite. On the other hand, TEFSMs are EFSMs where the time at which actions occur is considered. Stimulating a TEFSM implies producing some inputs at some times. Variables are also used by TEFSMs, but some of them may denote *clocks*, that is, they denote the time elapsed since some previous transition was taken.[15] We will assume that the time is discrete and only executions up to a given number of time units are considered.

Let C^{FSM}_{klm} represent the set of all deterministic FSMs having at most $k \in \mathbb{N}$ states, $l \in \mathbb{N}$ inputs, and $m \in \mathbb{N}$ outputs (that is, following the notation used in previous examples, we have $|I'| = l$ and $|O'| = m$). Besides, let $\mathcal{T}_{FSM} = \{(C^{FSM}_{klm}, \{f\}) \mid k,l,m \in \mathbb{N}, f \in C^{FSM}_{klm}\}$. Similarly, let C^{EFSM}_{klmpq} represent the set of all deterministic EFSMs with the same meaning of k,l,m as before and having at most $p \in \mathbb{N}$ variables, where each variable can take up to $q \in \mathbb{N}$ different values, and let $\mathcal{T}_{EFSM} = \{(C^{EFSM}_{klmpq}, \{f\}) \mid k,l,m,p,q \in \mathbb{N}, f \in C^{EFSM}_{klmpq}\}$. Finally, let $C^{TEFSM}_{klmpqrs}$ represent the set of all deterministic TEFSMs with the same meaning of k,l,m,p,q as before, where the first r variables denote clocks and executions can take up to s time units, and let $\mathcal{T}_{TEFSM} = \{(C^{TEFSM}_{klmpqrs}, \{f\}) \mid k,l,m,p,q,r,s \in \mathbb{N}, f \in C^{TEFSM}_{klmpqrs}\}$. Let us compare testing problems $\mathcal{T}_{FSM}$, $\mathcal{T}_{EFSM}$, and $\mathcal{T}_{TEFSM}$.

FSMs can be seen as EFSMs where all transition guards are enabled. Thus, it will be easy to check $\mathcal{T}_{FSM} \leq_F \mathcal{T}_{EFSM}$. Given a pair $a = (C^{FSM}_{klm}, \{f\}) \in \mathcal{T}_{FSM}$, let us consider the pair $b = (C^{EFSM}_{klm00}, \{f\}) \in \mathcal{T}_{EFSM}$. Let us note that functions in C^{FSM}_{klm} are the same functions as those in C^{EFSM}_{klm00}, that is, $C^{FSM}_{klm} = C^{EFSM}_{klm00}$, so $a = b$ and any complete test suite for b is also a complete test suite for a. Thus, functions e and t required by Definition 12 to transform pairs from $\mathcal{T}_{FSM}$ into $\mathcal{T}_{EFSM}$ and transform test suites for $\mathcal{T}_{EFSM}$ into test suites for $\mathcal{T}_{FSM}$, respectively, can be defined such that $e((C^{FSM}_{klm}, \{f\})) = (C^{EFSM}_{klm00}, \{f\})$ and t is the identity function.[16] Given $a \in \mathcal{T}_{FSM}$, if $\mathcal{I}$ is a complete test suite for $e(a)$ then $t(\mathcal{I})$ is a complete test suite for a, so $\mathcal{T}_{FSM} \leq_F \mathcal{T}_{EFSM}$. On the other hand, EFSMs can be seen as TEFSMs where no variable represents a clock. Let us note that, contrarily to FSMs or EFSMs, stimulating a TEFSM does not consist in producing a sequence of inputs (buttons), but it consists in producing a sequence of inputs at some specific times, that is, a test for a TEFSM is a sequence of pairs (i,d) where i is an input and d is the time when i is produced. Let us consider a function e such that $e((C^{EFSM}_{klmpq}, \{f\})) = (C^{TEFSM}_{klmpq00}, \{f'\})$ with $f'((i_1,d_1)\cdots(i_n,d_n)) = f(i_1\cdots i_n)$. Given $a \in \mathcal{T}_{EFSM}$, a test suite for $e(a)$ consists in a set of sequences of pairs (i,d) of inputs and times. However, since $e(a)$ follows the form $(C^{TEFSM}_{klmpq00}, \{f'\})$, times in these sequences are irrelevant. Given a complete test suite for $e(a)$, it is straightforward to see that the set of sequences of inputs of that suite *without* the times is complete for a. Thus, we can define t as follows: $t(\mathcal{I}) = \{(i_1\cdots i_n) \mid ((i_1,d_1)\cdots(i_n,d_n)) \in \mathcal{I}\}$. By considering these definitions of e and t, the requirement imposed by Definition 12 is fulfilled: If $\mathcal{I}$ is a complete test suite for $e(a)$ then $t(\mathcal{I})$ is a complete test suite for a. Thus we conclude $\mathcal{T}_{EFSM} \leq_F \mathcal{T}_{TEFSM}$. By the transitivity of $\leq_F$, we also have $\mathcal{T}_{FSM} \leq_F \mathcal{T}_{TEFSM}$. Thus, the problem of *completely* testing (i.e. testing up to completeness) FSMs can be easily transformed into the problem of completely testing TEFSMs, as one would expect.

More interestingly, we also have $\mathcal{T}_{EFSM} \leq_F \mathcal{T}_{FSM}$ and $\mathcal{T}_{TEFSM} \leq_F \mathcal{T}_{EFSM}$. Let us note that any EFSM where the number of variables is finite, and variables can take a finite number of values, can be *unfolded* into an equivalent FSM where each state represents a combination of EFSM state and variable values. Let $e((C^{EFSM}_{klmpq}, \{f\})) = (C^{FSM}_{(k*p*q)\ lm}, \{f\})$. Given $a \in \mathcal{T}_{EFSM}$, a complete test suite for $e(a)$ is a complete test suite for a (note that $C^{EFSM}_{klmpq} = C^{FSM}_{(k*p*q)\ lm}$, so $e(a) = a$). Thus, if t is the identity function then e and t fulfill the requirement of Definition 12 and we have $\mathcal{T}_{EFSM} \leq_F \mathcal{T}_{FSM}$.

On the other hand, let us note that a TEFSM where executions are constrained to take at most s time units can be simulated by an EFSM. We can transform the TEFSM as follows. Each transition labeled by i in the TEFSM is transformed into up to s transitions in the EFSM, one for each pair (i,d) where $0 \leq d \leq s$ represents a time. Actually, each (i,d) is considered to be a *single* EFSM input. Due to the s limit, the number of EFSM inputs and transitions is finite. Besides, each transition labeled by (i,d) explicitly updates the value of all EFSM variables representing the TEFSM clocks according to d. Let $e((C^{TEFSM}_{klmpqrs}, \{f\})) = (C^{EFSM}_{k\ (l*s)\ mp\ (max(q,s))}, \{f\})$, and let t map TEFSM sequences of inputs and times in such a way that each pair (i,d) in a sequence is transformed into the (single) input representing that pair (i,d) in the EFSM. Given $a \in C^{TEFSM}_{klmpqrs}$, by construction we have that, if $\mathcal{I}$ is a complete test suite for $e(a)$, then $t(\mathcal{I})$ is a complete test suite for a, so we conclude $\mathcal{T}_{TEFSM} \leq_F \mathcal{T}_{EFSM}$. By the transitivity of $\leq_F$, we have $\mathcal{T}_{TEFSM} \leq_F \mathcal{T}_{FSM}$, that is, the problem of finding a complete test suite for a bounded TEFSM can be transformed into the problem of finding a complete test suite for a FSM having less than a given number of states. As we saw in Example 6, for all $a = (C^{FSM}_{klm}, \{f\}) \in \mathcal{T}_{FSM}$ we have $a \in$ `Class I`, so $\mathcal{T}_{FSM} \subseteq$ `Class I`. By Theorem 3 (c) we conclude $\mathcal{T}_{EFSM} \subseteq$ `Class I` and $\mathcal{T}_{TEFSM} \subseteq$ `Class I`.

The previous examples of this case study lie in mapping computation formalism, which was possible in particular due to the finiteness of all involved computation formalisms. Next, an example of reduction where a mapping between computation formalisms is impossible (because they trivially have different computing power)

15. TEFSMs can be thought as a kind of *timed automata*.

16. Let us note that e is also the identity function.

is presented. Moreover, despite both computation formalisms are infinite, we will see that they lie in `Class I`. Other kinds of examples involving infinite computation formalisms that cannot be transformed into each other will be considered in Case Study 5.4.

Let C_{TM} represent all deterministic terminating Turing Machines from $\{0,1\}^*$ to $\{yes, no\}$, and C_{FA} represent all deterministic finite automata with the same type.

Let $\mathcal{T}_{TM}$ consist of all pairs where we have to distinguish a Turing Machine M from all Turing Machines answering as M for at most $k \in \mathbb{N}$ inputs. Formally, let $\mathcal{T}_{TM}$ be defined as follows: $\mathcal{T}_{TM} = \{(\{f\} \cup C_{TM}^{f,k}, \{f\}) | f \in C_{TM}, k \in \mathbb{N}\}$ where $f' \in C_{TM}^{f,k}$ iff f' gives the same answer as f for at most k words.

Similarly, let $\mathcal{T}_{FA} = \{(\{f\} \cup C_{FA}^{f,k}, \{f\}) | f \in C_{FA}, k \in \mathbb{N}\}$ where $f \in C_{FA}^{f,k}$ iff f' gives the same answer as f for at most k words. Let us note that any pair in $\mathcal{T}_{TM}$ or $\mathcal{T}_{FA}$ is uniquely characterized by f and k.

Next we will show $\mathcal{T}_{TM} \leq_F \mathcal{T}_{FA}$. In particular, functions e and t considered in Definition 12 can be defined as follows. First, $e : \mathcal{T}_{TM} \longrightarrow \mathcal{T}_{FA}$ is such that $e((\{f\} \cup C_{TM}^{f,k}, \{f\})) = (\{f'\} \cup C_{FA}^{f',k}, \{f'\})$, where f' is *any* function in C_{FA} (say, the one answering *yes* for all inputs), and $t : 2^{\{0,1\}^*} \longrightarrow 2^{\{0,1\}^*}$ is the identity function. Let $p = (\{f\} \cup C_{TM}^{f,k}, \{f\}) \in \mathcal{T}_{TM}$. We have that $\mathcal{I}$ is a complete test suite for $e(p) = (\{f'\} \cup C_{FA}^{f',k}, \{f'\})$ only if $\mathcal{I}$ consists of (any) $k+1$ or more different inputs. If it is so then $t(\mathcal{I}) = \mathcal{I}$ is also complete for $p = (\{f\} \cup C_{TM}^{f,k}, \{f\}) \in \mathcal{T}_{TM}$. So, $\mathcal{T}_{TM} \leq_F \mathcal{T}_{FA}$. For all $q = (\{f\} \cup C_{FA}^{f,k}, \{f\}) \in \mathcal{T}_{FA}$, we have that any set of $k+1 \in \mathbb{N}$ inputs is a complete test suite for q, so $\mathcal{T}_{FA} \subseteq$ `Class I`. By Theorem 3 (c) we conclude $\mathcal{T}_{TM} \subseteq$ `Class I`. By using similar arguments we have $\mathcal{T}_{FA} \leq_F \mathcal{T}_{TM}$.

5.4 Case study: Increasing Continuous Magnitude Machines

In this case study, we introduce a model of machine that generalizes (a kind of) temporal systems, and we use testing reductions to find ways to test several variants of this model. Actually, some reductions will show interesting methods to test them and some (not so expected) inclusion in `Class I` (in particular, that of testing problem $\mathcal{T}_V$). In addition, we will use these notions to briefly discuss the practical applicability of the proposed methods to test a car control device.

Let $F_{k,I,O}$ consist of all deterministic FSMs with no more that $k \in \mathbb{N}$ states, the finite set of inputs I, and the finite set of inputs O. Let $\mathcal{T}_F = \{(F_{k,I,O}, \{f\}) \mid k \in \mathbb{N} \wedge I, O$ are finite sets $\wedge f \in F_{k,I,O}\}$. We already know that all pairs in $\mathcal{T}_F$ belong to `Class I` (see Example 3).

Let us introduce a new state machine which we will call *Increasing Continuous Magnitude machine* (ICM hereafter). This deterministic machine has a finite number of states and transitions as a standard FSM, though it has some new elements. In addition to standard inputs, it also receives a (never decreasing) continuous signal. For instance, this magnitude can denote the number of kilometers ran by a car, the number of gas liters consumed by the car, or even the elapsed time. A *magnitude increment counter* accounts the increment of the magnitude since it was reset to 0 for the last time. There are three kinds of transitions in an ICM. First, we have the standard FSM transitions labeled by an input/output pair (called *standard i/o* transitions). Second, we have transitions like the former ones, though they also reset the magnitude counter to 0 when triggered (*resetting i/o* transitions). Finally, we also have transitions which are taken when the magnitude increment counter reaches some value $r \in \mathbb{R}^+$ specified in the transition. After the transition is (silently) taken, the counter is also reset to 0. These transitions will be called *triggering* transitions. We will assume that all states have exactly one outgoing triggering transition (note that we can still represent a state that cannot be abandoned without receiving any input by making its mandatory triggering transition lead to the same state). An interaction with an ICM consists in a sequence where standard inputs from a finite set I can interleave with actions of the form "increase the magnitude by x units" with $x \in \mathbb{R}^+$, in any order. The ICM replies to inputs with *outputs*, whereas magnitude increments produce no visible answer.

Let $W_{k,I,O}$ be the set of all functions which denote the behavior of some ICM with at most k states and sets of inputs and outputs I and O, respectively. Let $f \in W_{k,I,O}$. It is easy to see that $(W_{k,I,O}, \{f\}) \notin$ `Class I`. Let us suppose that we have observed the reaction of the IUT in these two scenarios: (a) when an input i is provided after increasing the magnitude by u_1 units; and (b) when it is provided after increasing it by u_2 units with $u_2 > u_1$. We know nothing about what would happen if i were given after u units with $u_2 > u > u_1$. In particular, the ICM could silently move to another state just before u, and then again just after u. Thus, similarly as in the pair (C_3, E_3) viewed in Example 3, *all* magnitude increments must be checked to get completeness. Hence $(W_{k,I,O}, \{f\}) \notin$ `Class I`.

However, we can enable finite completeness in many alternative related scenarios. For instance, let us suppose that we can *see* triggering transitions when they happen. Then, we can check the value of the magnitude counter at that exact moment when they are taken. Let us assume that we extend the original ICM set of inputs I with a new input called *'increase,'* meaning "*increase the magnitude until a triggering transition is taken.*" Also, the original ICM set of outputs O is extended with a finite set of outputs *'trigger at x'* with $x \in \mathbb{R}^+$, meaning *"a trigger transition was taken at magnitude counter value x."* Let I' and O' be the resulting extended sets (note that O' is infinite). Let $W'_{k,I',O'}$ denote the set of ICM variants modified like this and having the (extended) set of inputs I', the (extended) set of outputs O', and having at most k states, and let $\mathcal{T}_{W'} = \{(W'_{k,I',O'}, \{f\}) \mid k \in \mathbb{N} \wedge I', O'$ are the described extensions for some finite

sets $I, O \ \wedge \ f \in W'_{k,I',O'}\}$. We have $\mathcal{T}_{W'} \leq_F \mathcal{T}_F$. In order to make this reduction, we just convert any pair $(W'_{k,I',O'}, \{f\}) \in \mathcal{T}_{W'}$ into a pair $(F_{k,I',O''}, \{f'\}) \in \mathcal{T}_F$. In that expression, f' represents an FSM where, contrarily to f, interactions consist in sequences of *only* I' inputs (note that f also reacts to sequences where I' inputs may interleave in any order with ICM actions of the form "increase the magnitude by x units" with $x \in \mathbb{R}^+$, which do not exist in f'). Besides, the set of outputs O'' consists of all outputs of O, together with all outputs of the form *'trigger at x'* where x is the label of some triggering transition of f, together with an additional output *'trigger at unknown value.'* All outputs of the form *'trigger at x'* given by functions in $W'_{k,I',O'}$ are translated into the *'trigger at unknown value'* output if x is not a triggering value of f. Note that, contrarily to O', O'' is a finite set. We can see that any complete test suite for $(F_{k,I',O''}, \{f'\})$ is a complete test suite for $(W'_{k,I',O'}, \{f\})$, because testing $(F_{k,I',O''}, \{f'\})$ up to completeness implies fully controlling the reactions to magnitude increments in $(W'_{k,I',O'}, \{f\})$. So, $\mathcal{T}_{W'} \leq_F \mathcal{T}_F$. Since $\mathcal{T}_F \subseteq$ `Class I`, we have $\mathcal{T}_{W'} \subseteq$ `Class I`.

Finite completeness can also be enabled if we assume that we know the finite set J of values $x \in \mathbb{R}^+$ that label triggering transitions in the IUT (or we know any finite superset of J). Let $W''_{k,I,O,J} \subseteq W_{k,I,O}$ denote all ICMs with at most k states, set of inputs I, and set of outputs O, where the set of values labeling all triggering transitions is a subset of J. Let $\mathcal{T}_{W''} = \{(W''_{k,I,O,J}, \{f\}) \mid k \in \mathbb{N} \ \wedge \ I, O$ are finite sets $\ \wedge$ all values in triggering transitions belong to $J \ \wedge \ f \in W''_{k,I,O,J}\}$. We have $\mathcal{T}_{W''} \leq_F \mathcal{T}_F$. In particular, we can convert any pair $(W''_{k,I,O,J}, \{f\}) \in \mathcal{T}_{W''}$ into a pair $(F_{k',I',O}, \{f'\}) \in \mathcal{T}_F$ where

- I' consists of all inputs of I together with a set of new inputs *'increase magnitude by x'* for all $x \in J$ (note that J is finite, so I' remains finite as well);
- f' represents an FSM that behaves (partially) as the ICM represented by f. In particular, FSM states are pairs (s, m) where s is a state of the ICM and m is a magnitude value which must be the addition of 0 or more values of J and must not be higher than max (where max is the highest value of J). In addition, $max + 1$ (meaning 'any value higher than m') is also an allowed value for m. Note that the set of allowed values of m is finite, so the set of states of the FSM is so as well. Transitions for each FSM state (s, m) are defined to simulate the corresponding ICM: s changes as defined by the corresponding standard or resetting i/o transition, and m is increased in *'increase magnitude by x'* inputs (with $x \in J$) by x units as long as $m+x \leq max$, else it is set to $max+1$. Besides, m turns to 0 in FSM transitions which denote an ICM resetting i/o or triggering transition; and
- k' equals k (the number of allowed states in ICMs represented by $W''_{k,I,O,J}$) multiplied by the number of possible values of m according to the previous explanation. Note that any ICM represented by $W''_{k,I,O,J}$ can be converted, according to the aforementioned construction, into an FSM with k' states.

Since all values labeling triggering transitions in the ICM representing the IUT belong to J (by hypothesis), we have that any complete test suite for $(F_{k',I',O}, \{f'\})$ is also a complete test suite for $(W''_{k,I,O,J}, \{f\})$ (any complete test suite for $(F_{k',I',O}, \{f'\})$ must check all FSM transitions, which by the way also implies checking all reactions of the original ICM to magnitude increments). We infer $\mathcal{T}_{W''} \leq_F \mathcal{T}_F$.

We modify the previous case to denote a similar scenario. Rather than the set J, we could just know the *greatest common divisor* gcd of all values in J, which makes sense if all real values in J turn out to be also naturals, or at least rational numbers. In addition, let us suppose that we also know an upper bound max on the highest value labeling a triggering transition in the IUT. Let $\mathcal{T}_{W'''}$ be this new testing problem. We can perform the same reduction into $\mathcal{T}_F$ as in the previous paragraph, though now ICMs are translated into FSMs where a single *'increase magnitude by x'* new input is added, where $x = gcd$. States have also the form (s, m), though now m can take any multiple of gcd from 0 to the lowest multiple being higher than max. We infer $\mathcal{T}_{W'''} \leq_F \mathcal{T}_F$, so $\mathcal{T}_{W'''}$ is also included in `Class I`. Note that this case is very similar to assuming that magnitudes can take only *natural* numbers and we know an upper bound of the maximum label of triggering transitions. In addition, let us consider the following problem variant. Let $\mathcal{T}_{W''''} \subseteq T_{W''}$ be the same problem as the problem $\mathcal{T}_{W''}$ we saw in the previous paragraph, though now all values in J are rational numbers. After proving $\mathcal{T}_{W''''} \subseteq$ `Class I` like proposed in this paragraph, proving that $\mathcal{T}_{W''''}$ is included in `Class I` is much easier than doing it by proving $\mathcal{T}_{W''} \subseteq$ `Class I`. Since all values of J are assumed to be rational values in $\mathcal{T}_{W''''}$, there exists a greatest common divisor for all of them, so the reduction $\mathcal{T}_{W''''} \leq_F \mathcal{T}_{W'''}$ is trivial. Since $\mathcal{T}_{W'''} \subseteq$ `Class I`, we deduce $\mathcal{T}_{W''''} \subseteq$ `Class I`.

Let us introduce a more interesting scenario that (somehow unexpectedly) enables finite completeness. Note that, even if the conditions required by $\mathcal{T}_{W'''}$ apply, the size of the complete test suite dramatically increases as the greatest common divisor decreases. For instance, if two ICM triggering transitions trigger at values 4703.2 and 4703.25, then all magnitude increments of 0.05 units up to (at least) 4703.25 have to be checked at each ICM state in order to get completeness. Next we present a scenario where this problem is avoided.

Let us discard all previously presented hypotheses and assume the following ones:

(a) All values labeling triggering transitions are higher than some given $l \in \mathbb{R}^+$.

(b) For some given $r \in \mathbb{R}^+$ with $2r < l$, any wrong output happening within less than r magnitude units away from another magnitude value where that output can be produced by the specification ICM will *not* be considered as a fault. In order to forbid cumulative small

deviations where none individually exceeds the limit, *total* magnitude increments from the execution beginning are considered. For instance, let us suppose that the sequence *"increase 5.3 units; give i_1 and receive o_1; increase 7.9 units; give i_2 and receive o_2"* cannot be produced by the specification ICM, but the sequence *"increase 5.7 units; give i_1 and receive o_1; increase 7.2 units; give i_2 and receive o_2"* can. If we set $r = 0.5$, then the former sequence is not considered as a fault, because the first output happens within the 0.5 units margin from 5.7, and the second one happens within the 0.5 units margin from $5.7+7.2 = 12.9$ (it happens at value $5.3 + 7.9 = 13.2$). However, if magnitude 7.9 is replaced by 7.0 in the former sequence, then the resulting sequence is considered wrong, because now o_2 happens 0.6 units away from its allowed value ($5.3 + 7.0 = 12.3$).

Let $V_{k,I,O,l}$ be the set of all functions denoting the behavior of some ICM with at most k states and sets of inputs and outputs I and O, respectively, where the values of all triggering transitions are higher than l. Let $f \in V_{k,I,O,l}$ and $V_{f,r} \subseteq V_{k,I,O,l}$ be the subset of all functions in $V_{k,I,O,l}$ which are correct with respect to f according the criterion (b). Let $\mathcal{T}_V = \{(V_{k,I,O,l}, V_{f,r}) \mid k \in \mathbb{N} \wedge l, r \in \mathbb{R}^+ \wedge r < l \wedge f \in V_{k,I,O,l} \wedge I, O$ are finite sets $\wedge\, V_{f,r} \subseteq V_{k,I,O,l}$ contains all functions being correct with respect to f according to (b)$\}$. Let us show that we have $\mathcal{T}_V \leq_F \mathcal{T}_F$, which by the way will prove $\mathcal{T}_V \subseteq$ `Class I`. We can transform each pair $(V_{k,I,O,l}, V_{f,r}) \in \mathcal{T}_V$ into a pair $(F_{k',I',O}, \{f'\}) \in \mathcal{T}_F$ where

- I' consists of all inputs of I together with the following new input: *'raise magnitude'*;
- f' is a function representing an FSM behaving as follows. Its states are pairs (s, m) where s is a state of the ICM and m is a magnitude value which can be any multiple of l from 0 to the highest multiple of l that is lower than max, where max is the maximum value labeling a triggering transition in the specification machine. Moreover, m can also be r and $x - r$ for all $x \in \mathbb{R}^+$ labeling a triggering transition of the specification machine. Note that s and m can take a finite set of values, so the resulting machine has a finite number of states indeed. Transitions between states (s, m) are defined to simulate the state and magnitude changes in ICM transitions. Let us suppose we are at the FSM state (s, m). Taking ICM standard i/o transitions modifies s according to the transition and leaves m unmodified, whereas ICM resetting i/o transitions modify s as specified by the transition and also set m to 0. ICM triggering transitions are discarded and replaced in the FSM by new transitions triggered by a new input in I' called *'raise magnitude'*. The input *'raise magnitude'* simulates, in the FSM, some (partial or total) increase of the magnitude towards the lowest or the highest bound allowed by the (unique) ICM triggering transition value at s. Let x be the triggering value at s, and let us assume that the ICM triggering transition at s leads to s'. The FSM input *'raise magnitude'* has the following effect:

  (i) If the current magnitude value m is the highest multiple of l being less than $x - r$, then the magnitude is raised to reach $x - r$. This means that the new state reached in the FSM is $(s, x - r)$.

  (ii) Else, if the current magnitude value m equals $x - r$, then the magnitude is raised to reach $x + r$ (note that this increment is lower than l because we required $2r < l$). This means that the new state reached in the FSM is (s', r) (note that it is not $(s', 0)$).

  (iii) Else it simulates that the magnitude is increased up to the next multiple of l. That is, the new state reached in the FSM is (s, m') where m' is the smallest multiple of l that is higher than m.
- $k' = k \cdot (\lceil \frac{max}{l} \rceil + k + 1)$. That is, k' equals k (the number of allowed states in ICMs represented by $V_{k,I,O,l}$) multiplied by an upper bound of the number of possible values of m as mentioned above. Note that any ICM represented by $V_{k,I,O,l}$ can be converted, according to the aforementioned construction, into an FSM with k' states.

Let us note that any complete test suite for $(F_{k',I',O}, \{f'\})$ must also be a complete test suite for $(V_{k,I,O,l}, V_{f,r})$ after all *'raise magnitude'* FSM inputs appearing in the complete test suite for FSMs are replaced by the corresponding magnitude increments they represent in each case according to cases (i), (ii), and (iii) presented above. That is, if a test suite can guarantee that an FSM behaves as the FSM resulting from the previous conversion, then the original ICM is correct as well. On the hand, by using only $+l$ (or less) magnitude increments, there is no risk of missing any triggering transition from the IUT: Since all triggering transitions trigger at increments strictly higher than l, no more than one triggering transition may happen between two consecutive multiples of l. Thus, there is no risk to leave any IUT state *unchecked*. On the other hand, by checking triggering values just r units before and after the required triggering value, we can check whether the behavior at both points corresponds to the behavior of the ICM at the state we are supposed to be at *just before the triggering transition* and the expected state *just after it*. If both behaviors are as expected, then the triggering must have happened between both values, as required by condition (b) introduced earlier. We conclude that the proposed reductions enables $\mathcal{T}_V \leq_F \mathcal{T}_F$, so we have $\mathcal{T}_V \subseteq$ `Class I`.

Let us suppose that we want to extend ICMs to make them depend on *two* independent magnitudes. Both magnitudes have their own counter, both can be independently reset in transitions, and both can launch their own independent triggering transitions when their counters reach some specific values. Also, let us assume that both may have their own independent l and r constants as defined in the previous case. The problem of testing ICMs with two magnitudes under these assumptions (say $\mathcal{T}_{V^2}$) can be reduced to the problem of testing ICMs as considered in the previous case ($\mathcal{T}_V$) by using almost

the same construction as in the reduction $\mathcal{T}_V \leq_F \mathcal{T}_F$ seen before. In particular, we get rid of one magnitude of the 2-magnitude ICM by codifying it into the FSM part (i.e. states and transitions) of the 1-magnitude ICM (again, we use the *'raise magnitude'* additional input for simulating increases on the removed magnitude). We conclude $\mathcal{T}_{V^2} \leq_F \mathcal{T}_V$. Let $\mathcal{T}_{V^n}$ be the problem of testing ICMs with n independent magnitudes. Similarly, it can be reduced to $\mathcal{T}_{V^{n-1}}$. Since we also have $\mathcal{T}_V \leq_F \mathcal{T}_F$ and $\mathcal{T}_F \subseteq$ `Class I`, by induction and the transitivity of $\leq_F$ we trivially infer that $\mathcal{T}_{V^n} \subseteq$ `Class I` for all $n \in \mathbb{N}$. In order to get the actual complete test suite derivation algorithm for $\mathcal{T}_{V^n}$ instances, we just have to compose all the reductions from each problem to the subsequent one, next apply a derivation algorithm for FSMs (e.g. test all sequences with $2n+1$ inputs, where n is the number of states), and next iteratively translate complete test suites back up to $\mathcal{T}_{V^n}$. In particular, we convert all *'raise magnitude'* inputs, used at each magnitude hiding, into the true magnitude increments corresponding at each state (s, m).

Let us discuss some practical implications of the previous testing methodology. First, note that the *time* can be considered as any other magnitude of an ICM. In order to increase this magnitude, we just have to let the time pass by. Second, let us address the following question: Can we use that previous testing methodology for $\mathcal{T}_{V^n}$, inferred by consecutively applying reductions, to test ICMs where magnitudes are *not* independent? Let us illustrate this issue with the following example. We want to test a car control device which depends on the increments of the number of traversed kilometers, the number of consumed gasoline liters, and the amount of elapsed time. We may argue that, when the device is deployed in an actual car, we could hardly observe kilometer increments without associated increments in time or in the consumed gasoline. Let us suppose that we can test the device *before* it is deployed in a car. In the laboratory, we are free to stimulate all IUT sensors (time, kilometers, gas consumption) independently from each other. Let us suppose that we apply the test suite derivation algorithm mentioned in the previous paragraph (in particular, the one for 3-magnitude ICMs) to test this IUT, and the IUT passes it. Since the derived test suite is complete, it means that the IUT is correct according to our criteria (a) and (b) mentioned earlier. Moreover, since the test suite discards *all* possible wrong IUTs according to these criteria, in particular it also discards all possible wrong IUTs where magnitudes increments are constrained to those that could really happen inside a car. For instance, interactions where every 1 kilometer we add 1 minute and 0.05 liters are *also* possible in an independent-magnitude setting, so they are also required to be right by a complete test suite. Thus, any possible wrong behavior, unleashed by a realistic after-deployment magnitude increment pattern, is also guaranteed not to happen if the test suite is passed (even if no test in the suite increases 1 kilometer every 1 minute and 0.05 liters), because the test suite is *complete*. Hence, if the IUT passes this laboratory test (developed for ICMs where magnitudes are independent), then it will also work in deployment conditions – as long as magnitude sensors also work well and the IUT logical definition does not "change" in deployment conditions (e.g. due to chip overheating caused by the engine). That is, the 3-independent-magnitudes testing methodology completely checks the *logic* of the IUT – also for the particular *logic* scenarios that may happen under deployment conditions.

Alternatively, we could study how to *explicitly* impose constraints between increments of different magnitudes in the testing reductions given before, so that the completeness is required *only* for those magnitude increments that can be produced after the real deployment. Note that, in this case, the new derived complete test suites would *not* be required to guarantee the correctness when we add 100 kilometers, 0 minutes, and 0 gas liters consumption, because this case would be considered as impossible under deployment conditions. Studying these alternative reductions is beyond the scope of this case study.

Let us note that all the reductions given in this case study could be easily adapted to the case where magnitudes are allowed to both increase and *decrease*, and triggering transitions can be launched by both increments and decrements. For instance, in the $\mathcal{T}_V \leq_F \mathcal{T}_F$ reduction given before, we would just have to consider both increments towards *increment triggering transitions* and decrements towards *decrement triggering transitions*. This way, adding e.g. a temperature increase/decrease factor to our car device example would be simple.

Finally, let us note that the $\mathcal{T}_{W'} \leq_F \mathcal{T}_F$ and $\mathcal{T}_V \leq_F \mathcal{T}_F$ reductions given in this example are particularly interesting because they are *asymmetric*: In both cases, the computation formalism of any pair in the origin testing problem is *infinite* (in particular, there are infinitely many deterministic ICMs with some given number of states *and* fulfilling the specific constraints considered in these cases, because both cases enable infinite possible values to label triggering transitions), but the computation formalism of all pairs of the destination problem in both cases, i.e. $\mathcal{T}_F$, is *finite* (the set of all behaviors which can be represented by an FSM with a given number of states is finite). In the rest of reductions studied in this case study, we have finiteness at both sides or infiniteness at both sides. Anyway, in all cases considered in the case study, the testing problems belong to `Class I`.

6 Proofs of Complexity Results

Proof of Theorem 1

First, let us prove `CS` $\in$ `NP`. Let us note that the representation size of C and E is in $\mathcal{O}(|C| \cdot |I| \cdot |O|)$ and the representation size of $\not\equiv$ is in $\mathcal{O}(|O|^2)$. Given C, E, I, O, $\not\equiv$, K as input of `CS`, we prove that checking whether

some given set of inputs $\mathcal{I} \subseteq I$ is such that $|\mathcal{I}| \leq K$ and $\mathcal{I}$ is a complete test suite for C and E requires polynomial time. Since $\mathcal{I} \subseteq I$, we calculate $|\mathcal{I}|$ in time $\mathcal{O}(|I|)$. Next we have to check that, for all pair (f, f') with $f \in E$ and $f' \in C \backslash E$, there exists some $i \in \mathcal{I}$ such that for all $o \in f(i)$ and $o' \in f'(i)$ we have $o \not\neq o'$. This can be done in time $\mathcal{O}(|C|^2 \cdot |I| \cdot |O|^2)$. Thus, CS $\in$ NP.

In order to prove CS $\in$ NP-complete, we polynomially reduce an NP-complete problem to CS. We will consider the *Set Cover* problem. This NP-complete is defined as follows: Given a set S, a collection J of subsets of S, and a natural number $K \in \mathbb{N}$, find a *set cover* J' for S (i.e., find a subset $J' \subseteq J$ such that every element in S belongs to at least one member of J') such that $|J'| \leq K$.[17]

Given a set $S = \{x_1, \ldots, x_n\}$ and a collection of sets $J = \{S_1, \ldots, S_k\}$ with $S_j \subseteq S$ for all $1 \leq j \leq k$, we will construct some C, E, I, O from them in polynomial time. The idea of the transformation from Set Cover to CS is the following: We construct an instance of CS where there are some correct functions and a *single* incorrect function; in particular, we will have $C = \{f_1, \ldots, f_n, f'\}$ and $E = \{f_1, \ldots, f_n\}$ (that is, f' is the only incorrect function). For all $1 \leq j \leq n$, the pair (f_j, f') with $f_j \in E$ and $f' \in C \backslash E$ will be used to *represent* the element x_j of the set S. Besides, each input of CS will represent a *set* of J. In particular, f_j and f' will be defined in such a way that (f_j, f') are distinguished by input i_l if and only if $x_j \in S_l$. Since each input represents a set and each pair of correct/incorrect functions represents an element, the task of finding K inputs such that all pairs of correct/incorrect functions are distinguished will be equivalent to finding K sets such that all elements of S are covered by them. Let us note that the set of outputs returned by a function for an input must include at least one element. In our case, each function will return at least a (unique) output for each input: For each function f_j and input i_l, we will consider $\alpha_l^j \in f_j(i_l)$ for some unique output α_l^j. We will use the trivial distinguishing relation, i.e. $o_1 \not\neq o_2$ iff $o_1 \neq o_2$. Thus, if only the unique outputs α_l^j were returned, all pairs (f_j, f') would be distinguished by all inputs. Following the proposed analogy between both problems, (f_j, f') should be distinguished by i_l only if $x_j \in S_l$. Thus, if $x_j \notin S_l$ then we must impose a condition to avoid that (f_j, f') is distinguished by i_l. In particular, we will consider $\beta_l \in f_j(i_l)$ and $\beta_l \in f'(i_l)$ for some output β_l. Since $f_j(i_l) \cap f'(i_l) \neq \varnothing$, the pair (f_j, f') is not distinguished by i_l. Hence, distinguishing (f_j, f') (i.e., *covering* the element x_j) will require to include another input in the test suite (i.e., another set in the collection).

We define $C, E, I, O, \not\neq$ from J, S as follows:

- $C = \{f_j = (outs_1^j, \ldots, outs_k^j) | 1 \leq j \leq n\} \bigcup \{f' = (outs_1', \ldots, outs_k')\}$
  where:
  - for all $1 \leq j \leq n$ and $1 \leq l \leq k$ we have $outs_l^j = \{\alpha_l^j\}$ if $x_j \in S_l$, else $outs_l^j = \{\alpha_l^j, \beta_l\}$,
  - for all $1 \leq l \leq k$ we have $outs_l' = \{\alpha_l', \beta_l\}$.
- $E = \{(outs_1^j, \ldots, outs_k^j) | (outs_1^j, \ldots, outs_k^j) \in C\}$
- $I = \{i_1, \ldots, i_k\}$
- $O = \{\alpha_l^j | 1 \leq j \leq n \ \wedge \ 1 \leq l \leq k\} \cup \{\alpha_l' | 1 \leq l \leq k\} \cup \{\beta_l | 1 \leq l \leq k\}$
- $\not\neq = \{(o_1, o_2) | o_1, o_2 \in O \ \wedge \ o_1 \neq o_2\}$

It is easy to check that the representation size of C and E is in $\mathcal{O}(|S| \cdot |J|)$ ($|S| + 1$ functions, $|J|$ inputs for each function, and one or two outputs for each input). There are $\mathcal{O}(|S|)$ inputs in I and $\mathcal{O}(|S| \cdot |J|)$ outputs in O, so the representation size of $\not\neq$ is in $\mathcal{O}(|S|^2 \cdot |J|^2)$. Thus, the constructed instance of CS has polynomial size.

We check that there exists $J' \subseteq J$ with $\bigcup_{S' \in J'} S' = S$ and $|J'| = K$ iff there exists a complete test suite $\mathcal{I}$ for C, E with $|\mathcal{I}| = K$:

$\Longrightarrow$: Let us suppose $J' = \{S_{a_1}, \ldots, S_{a_K}\}$ and $\bigcup_{S' \in J'} S' = S$. We check that $\mathcal{I} = \{i_{a_1}, \ldots, i_{a_K}\}$ is a complete test suite for C, E. Let us suppose that it is not. Then, there exists a pair (f_j, f') with $f_j \in E$ and $f' \in C \backslash E$ such that for all $i_l \in \{i_{a_1}, \ldots, i_{a_K}\}$ we have $f_j(i_l) \cap f'(i_l) \neq \varnothing$. By the definition of C and E, if $f_j(i_l)$ and $f'(i_l)$ share some element then this element must be β_l. So, for all $i_l \in \{i_{a_1}, \ldots, i_{a_K}\}$ we have $f_j(i_l) \cap f'(i_l) = \{\beta_l\}$. By the construction of C and E from J and S, if $\beta_l \in f_j(i_l)$ then $x_j \notin S_l$. Thus, $x_j \notin S_l$ for all $S_l \in \{S_{a_1}, \ldots, S_{a_K}\} = J'$. Therefore, $\bigcup_{S' \in J'} S' \neq S$ and we get a contradiction.

$\Longleftarrow$: We assume that $\mathcal{I} = \{i_{a_1}, \ldots, i_{a_K}\}$ is a complete test suite for C, E, and we check that $J' = \{S_{a_1}, \ldots, S_{a_K}\}$ fulfills the property $\bigcup_{S' \in J'} S' = S$. Let us suppose it does not. Then, there exists $x_j \in S$ such that for all $S_l \in J'$ we have $x_j \notin S_l$. By the construction of C and E from J and S, this implies that for all $i_l \in \{i_{a_1}, \ldots, i_{a_K}\}$ we have $\beta_l \in f_j(i_l)$ and $\beta_l \in f'(i_l)$. Thus, no input of $\mathcal{I}$ allows to distinguish the pair (f_j, f') and $\mathcal{I}$ is not a complete test suite for C, E. Hence, we have a contradiction.

□

Proof of Theorem 2

(a): We prove FR $\in$ P. From an instance of FR, let us construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows: $\mathcal{V} = C$, and $(f_1, f_2) \in \mathcal{E}$ for all $f_1 \in E$ and $f_2 \in C \backslash E$ such that di$(f_1, f_2, \mathcal{I})$ does *not* hold. By Lemma 3, a set $R \subseteq C$ is such that $\mathcal{I}$ is a complete test suite for $(C \backslash R, E \backslash R)$ iff $f_1 \in R$ or $f_2 \in R$ for all $f_1 \in E$ and $f_2 \in C \backslash E$ such that di$(f_1, f_2, \mathcal{I})$ does not hold. Since vertexes in $\mathcal{V}$ are functions in C and edges in $\mathcal{E}$ are pairs of correct/incorrect functions that are not distinguished by $\mathcal{I}$, the problem of finding a set $R \subseteq C$ with $|R| \leq K$ such that $\mathcal{I}$ is a complete test suite for $(C \backslash R, E \backslash R)$ is equivalent to removing K or less vertexes from $\mathcal{G}$ in such a way that all edges are lost. That is, FR is equivalent to finding a subset of vertexes $\mathcal{V}' \subseteq \mathcal{V}$ such that $|\mathcal{V}'| \leq K$

17. We could consider another problem, such as the *Minimum Test Collection* problem. Despite its name, this problem is very different to CS and thus is not a good choice to construct a polynomial reduction from an NP-complete problem to CS. In this problem, a collection of sets is selected in such a way that, for all pair of elements, there is a set including only one of them.

and the subgraph $\mathcal{G}'$ of $\mathcal{G}$ induced by $\mathcal{V}\backslash\mathcal{V}'$ has no edges, that is, $\{(v,v')|(v,v') \in \mathcal{E}, \{v,v'\} \subseteq \mathcal{V}\backslash\mathcal{V}'\} = \varnothing$. Let us consider the *Vertex Cover* problem, which is stated as follows: Given a graph, find a set of K or less vertexes such that, for each edge, at least one endpoint of the edge is included in the set. Since solving FR requires that, for each edge, at least one of the endpoints of the edge is removed, we can solve FR in terms of the Vertex Cover by considering that the vertex cover to be found represents the set of vertexes (functions) to be *removed* (i.e. R). In particular, we can do as follows: From an FR problem instance, we construct the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as defined before and next we find a vertex cover Q of size K. Then, we have $R = Q$.

Though the *Vertex Cover* problem is NP-complete in general graphs, König's theorem [4] states that it is equivalent to the *Matching* problem if the graph is *bipartite* (and $\mathcal{G}$ is so). The Matching problem in bipartite graphs is defined as follows: Given a bipartite graph, find a set of K of more edges such that no vertex of the graph is at the endpoint of more than one edge in the set. By König's Theorem, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. Moreover, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, both problems are solved in time $\mathcal{O}(\sqrt{|\mathcal{V}|} \cdot |\mathcal{E}|)$ by the Hopcroft-Karp algorithm [20]. Hence, FR and the corresponding optimization problem of finding the *minimum* function removal can be solved by constructing $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and next running that algorithm for $\mathcal{G}$. Identifying all pairs (f_1, f_2) such that $\mathtt{di}\,(f_1, f_2, \mathcal{I})$ does not hold requires that, for all $f \in E$ and $f' \in C\backslash E$, we check whether for all input $i \in \mathcal{I}$ there exist $o \in f(i)$ and $o' \in f'(i)$ such that $o \mathrel{\mathring{=}} o'$. Thus, the cost of constructing $\mathcal{G}$ is in $\mathcal{O}(|C|^2 \cdot |\mathcal{I}| \cdot |O|^2)$. The cost of executing the Hopcroft-Karp algorithm is, in terms of the FR problem instance, in $\mathcal{O}(\sqrt{|C|} \cdot |C|^2) = \mathcal{O}(|C|^{5/2})$. Thus, FR (and its corresponding optimization problem) can be solved in time $\mathcal{O}(|C|^{5/2} + |C|^2 \cdot |\mathcal{I}| \cdot |O|^2)$, and we conclude $\mathtt{FR} \in \mathtt{P}$.

(b): We prove $\mathtt{FH} \in \mathtt{NP\text{-}complete}$. First we prove $\mathtt{FH} \in \mathtt{NP}$. Given C, E, I, O, $\not\mathrel{\mathring{=}}$, $\mathcal{I}$, $\mathcal{H}$, K as input of FH, we prove that we can check in polynomial time whether some given $R \subseteq \mathcal{H}$ is such that $|\bigcup_{H\in R} H| \leq K$ and $\mathcal{I}$ is a complete test suite for $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$. Since $R \subseteq \mathcal{H}$ and for all $H \in R$ we have $H \subseteq C$, we can calculate $|\bigcup_{H\in R} H|$ in time $\mathcal{O}(|\mathcal{H}| \cdot |C|^2)$. Subtracting $\bigcup_{H\in R} H$ from C and E requires that, for all $H \in R$, we remove all function $f \in H$ from these sets. Since for all $H \in R$ we have $H \subseteq C$, these subtractions can be done in time $\mathcal{O}(|\mathcal{H}| \cdot |C|^2)$. After making these subtractions, we have to check that, for all pair (f, f') with $f \in E\backslash(\bigcup_{H\in R} H)$ and $f' \in (C\backslash(\bigcup_{H\in R} H))\backslash(E\backslash(\bigcup_{H\in R} H))$, there exists some $i \in \mathcal{I}$ such that for all $o \in f(i)$ and $o' \in f'(i)$ we have $o \not\mathrel{\mathring{=}} o'$. Since $|E\backslash(\bigcup_{H\in R} H)| \leq |C|$ and $|(C\backslash(\bigcup_{H\in R} H))\backslash(E\backslash(\bigcup_{H\in R} H))| \leq |C|$, this can be done in time $\mathcal{O}(|C|^2 \cdot |I| \cdot |O|^2)$. Thus, checking whether some given $R \in \mathcal{H}$ fulfills the required conditions can be done in polynomial time. Hence, $\mathtt{CS} \in \mathtt{NP}$.

In order to prove $\mathtt{FH} \in \mathtt{NP\text{-}complete}$, we polynomially reduce an NP-complete problem, 3-SAT, to FH. Next we introduce some notation related to 3-SAT. This problem is stated as follows: Given a propositional logic formula φ expressed in conjunctive normal form where each disjunctive clause has at most 3 literals, is there any valuation ν satisfying φ? Let $\varphi \equiv (l_1^1 \vee l_2^1 \vee l_3^1) \wedge \ldots \wedge (l_1^n \vee l_2^n \vee l_3^n)$ be an input for 3-SAT. We denote by $\mathtt{props}(\varphi) = \{p_1, \ldots, p_m\}$ the set of propositional symbols appearing in φ. We denote the i-th disjunctive clause of φ by c_i, that is, $c_i \equiv l_{i1} \vee l_{i2} \vee l_{i3}$. A valuation ν is a function $\nu : \mathtt{props}(\varphi) \longrightarrow \{\top, \bot\}$. We say that ν satisfies a clause c_i if it makes true at least one literal of c_i. We say that ν satisfies φ if it makes true all clauses c_i of φ.

Let $\varphi \equiv (l_1^1 \vee l_2^1 \vee l_3^1) \wedge \ldots \wedge (l_1^n \vee l_2^n \vee l_3^n)$. We construct in polynomial time an instance $(C, E, I, O, \not\mathrel{\mathring{=}}, \mathcal{I}, \mathcal{H}, K)$ of the FH problem such that there is a set of hypotheses $R \subseteq \mathcal{H}$ with $|\bigcup_{H\in R} H| \leq K$ such that $\mathcal{I}$ is a complete test suite for $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$ if and only if φ is satisfiable. Before formally presenting this construction, we give an intuitive explanation. We will construct an FH instance where, for each literal l_k^j of φ, there will be one function in E and another function in $C\backslash E$. The test suite $\mathcal{I}$ will consist of a single input i, and the answers of each function f to input i will be defined in such a way that some specific pairs of functions will not be distinguishable by i. Each function will answer some output α that is given *only* by that function. Thus, if this were the only output produced by a function then this function would be distinguishable from any other function (as we will consider the trivial distinguishing relation: $o_1 \not\mathrel{\mathring{=}} o_2$ iff $o_1 \neq o_2$). However, if we want to make two functions f_1, f_2 not to be distinguished by i, then we will add some output β to the answer of *both* functions to i. In this way, $\mathtt{di}\,(f_1, f_2, \mathcal{I})$ will not hold. Thus, we can make any pair of functions (un-)distinguishable as required for our construction.

In FH, the set of functions to be removed must be the union of some hypotheses (i.e. sets of functions) from some set $\mathcal{H}$ of available hypotheses. For each literal l_k^j, we will have in $\mathcal{H}$ a hypothesis consisting of the function representing l_k^j in E and the function representing the same literal in $C\backslash E$, and there will not be any other hypotheses in $\mathcal{H}$. This guarantees that the removal of functions in E and $C\backslash E$ will be *symmetric*: The only way to remove a function representing some literal in E or $C\backslash E$ consists in removing the function representing the same literal in $C\backslash E$ and E, respectively.

There are two kinds of pairs of functions such that their behavior will be defined to make them undistinguishable. On the one hand, pairs of functions representing opposite literals (e.g. p and $\neg p$) in E and $C\backslash E$ will not be distinguishable. Given a solution to FH, we will consider that *non*-removed functions implicitly denote a *valuation* of the propositional symbols. For instance, if a function representing $\neg p$ is not removed then we

are implicitly considering $\nu(p) = \bot$; if neither p nor $\neg p$ are removed then $\nu(p)$ can be arbitrarily set to $\top$ or $\bot$. Thus, for each function in E and each function in $C\backslash E$ representing opposite literals, at least one of them must be removed. By the symmetry of E and $C\backslash E$, both E and $C\backslash E$ will represent the same valuation indeed. On the other hand, for each function representing a literal l_j^k in E (respectively, in $C\backslash E$), this function will not be distinguishable from the functions representing the *other* two literals of the *same* clause in $C\backslash E$ (resp. E). Due to the definition of the hypotheses set $\mathcal{H}$, this guarantees that, for each clause, at least four out of the six functions representing the literals of each clause (three in E and three in $C\backslash E$) will be removed (and, if two functions remain, then they will be the representation of the same literal in E and $C\backslash E$). Thus, for each FH solution, at least $4 \cdot n$ functions must be removed (recall that φ has n clauses). Actually, if *only* $4 \cdot n$ functions are removed, then it means that one literal of each clause in E (resp. $C\backslash E$) is made true. Moreover, literals made true in $C\backslash E$ (resp. E) do not contradict these literals, because pairs of contradicting functions are undistinguishable and thus at least one function of each pair is removed. By the symmetry, this means that true literals in E (resp. $C\backslash E$) do not contradict true literals in the *same* set. Thus, if at most $K = 4 \cdot n$ functions are removed in FH then φ must be satisfiable.

Next we formally construct an FH instance $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K)$ from the 3-SAT instance $\varphi \equiv (l_1^1 \vee l_2^1 \vee l_3^1) \wedge \ldots \wedge (l_1^n \vee l_2^n \vee l_3^n)$, where we consider $\texttt{props}(\varphi) = \{p_1, \ldots, p_m\}$. Each literal l_k^j will be represented by a function $(outs_k^j) \in E$ and $(outs_k'^j) \in C\backslash E$ including the (unique) output symbol α_k^j and $\alpha_k'^j$, respectively. Two functions denoting opposite literals in E and $C\backslash E$ will be made undistinguishable by including the same output β_y^q, where q identifies the propositional symbol and $y \in \{\top, \bot\}$. A function from E (resp. $C\backslash E$) will be made undistinguishable from the functions denoting the other literals from the same clause in $C\backslash E$ (resp. E) by including β_k^j (resp. $\beta_k'^j$).

- $C = \{(outs_k^j) | 1 \leq j \leq n, 1 \leq k \leq 3\} \bigcup \{(outs_k'^j) | 1 \leq j \leq n, 1 \leq k \leq 3\}$
  where:
  - for all $1 \leq j \leq n$, $1 \leq k \leq 3$ we have $outs_k^j = \{\alpha_k^j, \beta_y^q, \beta_k^j, \beta_{k_1}'^j, \beta_{k_2}'^j\}$ where $l_k^j \equiv p_q$ or $l_k^j \equiv \neg p_q$ (in the first case, $y = \top$ else $y = \bot$) and $\{k_1, k_2\} = \{1, 2, 3\}\backslash\{k\}$.
  - for all $1 \leq j \leq n$, $1 \leq k \leq 3$ we have $outs_k'^j = \{\alpha_k'^j, \beta_y^q, \beta_k'^j, \beta_{k_1}^j, \beta_{k_2}^j\}$ where $l_k^j \equiv p_q$ or $l_k^j \equiv \neg p_q$ (in the first case, $y = \bot$ else $y = \top$) and $\{k_1, k_2\} = \{1, 2, 3\}\backslash\{k\}$.
- $E = \{(outs_k^j) | (outs_k^j) \in C\}$
- $I = \{i\}$
- $O = \{\alpha_k^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup \{\alpha_k'^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup \{\beta_k^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup \{\beta_k'^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup \{\beta_y^q | 1 \leq q \leq m, y \in \{\top, \bot\}\}$
- $\not\approx = \{(o_1, o_2) | o_1, o_2 \in O \wedge o_1 \neq o_2\}$
- $\mathcal{I} = \{i\}$
- $\mathcal{H} = \{\{(outs_k^j), (outs_k'^j)\} | 1 \leq j \leq n, 1 \leq k \leq 3\}$
- $K = 4 \cdot n$

We show that the construction of $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K)$ from φ requires polynomial time with respect to the size of φ. There are $6 \cdot n$ functions in C, and for each of them the answer to a single input (i) is defined. This answer consists of 5 outputs. Besides, there are $12 \cdot n + 2 \cdot m$ outputs in O. Even if $\not\approx$ is extensionally defined, we have $|\not\approx| = |O|^2$, and we have $|\mathcal{H}| = |C|/2$. In conclusion, $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K)$ can be constructed from φ in polynomial time.

Let us show that the answer of FH to $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K)$ is *yes* iff φ is satisfiable.

$\Longrightarrow$: Let us suppose that there is a set $R \subseteq \mathcal{H}$ with $|\bigcup_{H \in \mathcal{H}} H| \leq K$ such that $\mathcal{I}$ is a complete test suite for $\big(C\backslash(\bigcup_{H \in R} H), E\backslash(\bigcup_{H \in R} H)\big)$. Then, by Lemma 3 we have that $f \in \bigcup_{H \in R} H$ or $f' \in \bigcup_{H \in R} H$ for all $f \in C$ and $f' \in C\backslash E$ such that $\texttt{di}(f, f', \mathcal{I})$ does not hold. Since functions representing *opposite* literals are made undistinguishable in the construction of the FH instance, for each pair of functions representing opposite literals, at least one of these functions must be included in $\bigcup_{H \in R} H$. By the definition of $\mathcal{H}$, all $H \in \mathcal{H}$ is such that $H = \{f, f'\}$ for some $f \in E$ and $f' \in C\backslash E$ representing the same literal of φ. Thus, for all $f \in E$ such that $f \in \bigcup_{H \in R} H$, the function f' representing the same literal as f in $C\backslash E$ is such that $f' \in \bigcup_{H \in R} H$, and vice versa. Since there is no pair of functions representing opposite literals formed by a function in E and a function in $C\backslash E$ in the set $C\backslash(\bigcup_{H \in R} H)$, by the symmetry between E and $C\backslash E$ there is no pair of opposite functions formed by two functions from the *same* set either E or $C\backslash E$. Thus, either literals represented by functions in $E\backslash(\bigcup_{H \in R})$ or literals represented in $(C\backslash E)\backslash(\bigcup_{H \in R} H)$ can be used to form a (consistent) valuation of propositional symbols as follows: We define a valuation ν where $\nu(p) = \top$ iff for some $1 \leq j \leq n$ and $1 \leq k \leq 3$ we have $l_k^j \equiv p$ and $(outs_k^j) \notin (\bigcup_{H \in R} H)$. Let us note that if ν makes true at least one literal from each clause then ν satisfies φ. By the construction of C and E, the function $f \in E$ representing a literal l is undistinguishable from the functions denoting the *other* two literals from the same clause in $C\backslash E$ and vice versa. Besides, $\bigcup_{H \in R} H$ cannot have a function $f \in E$ without having the function $f' \in C\backslash E$ representing the same literal in $C\backslash E$, and vice versa. Thus at most two functions, out of the six functions denoting the literals of a clause in E and $C\backslash E$, can be in $C\backslash(\bigcup_{H \in R} H)$. Consequently, the condition $|\bigcup_{H \in \mathcal{H}} H| \leq K = 4 \cdot n$ can be achieved only if, from

the six functions denoting each clause, exactly four of them are in $\bigcup_{H\in R} H$. Since we have $|\bigcup_{H\in\mathcal{H}} H| \leq K$ indeed, the valuation ν, which satisfies all literals not in $\bigcup_{H\in R} H$, satisfies at least one of the literals from each clause. Therefore, ν satisfies φ.

$\Longleftarrow$: Let us suppose that φ is satisfiable. We show that there exists $R \subseteq \mathcal{H}$ such that $|\bigcup_{H\in\mathcal{H}} H| \leq K$ and $\mathcal{I}$ is a complete test suite for the pair $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$. Let ν be a valuation of $\texttt{props}(\varphi)$ satisfying φ. Let $Q \subseteq C$ be such that, for all literal $c_j \equiv l_1^j \vee l_2^j \vee l_3^j$ of φ, we have

- $(outs_k^j), (outs_k'^j) \in Q$ for some $1 \leq k \leq 3$ such that either (a) for some q we have $l_k^j \equiv p_q$ and $\nu(p_q) = \top$; or (b) for some q we have $l_k^j \equiv \neg p_q$ and $\nu(p_q) = \bot$ (since ν satisfies $c_j \equiv l_1^j \vee l_2^j \vee l_3^j$, there exists such k)
- $(outs_{k_1}^j), (outs_{k_1}'^j), (outs_{k_2}^j), (outs_{k_2}'^j) \notin Q$, where $\{k_1, k_2\} = \{1,2,3\}\backslash\{k\}$.

Let us consider $R = \{H | H \in \mathcal{H}, H \cap Q = \varnothing\}$. Let us recall that, by the construction of $\mathcal{H}$, we have $\mathcal{H} = \{\{(outs_k^j), (outs_k'^j)\} | 1 \leq j \leq n, 1 \leq k \leq 3\}$. Therefore, we have $Q = C\backslash(\bigcup_{H\in R} H)$ and $|\bigcup_{H\in R} H| = 4 \cdot n$. By Lemma 3, if for all $f_1 \in E$ and $f_2 \in C\backslash E$ such that $\texttt{di}(f_1, f_2, \mathcal{I})$ does not hold we have either $f_1 \in \bigcup_{H\in R} H$ or $f_2 \in \bigcup_{H\in R} H$, then $\mathcal{I}$ is a complete test suite for $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$. By the construction of E and $C\backslash E$, a pair of functions is undistinguishable if either (a) they represent opposite literals or (b) they represent different literals from the same clause. By the construction of Q from ν, Q does not have a pair of functions representing opposite literals, and it does not have a pair of functions representing different literals from the same clause. Thus, there do not exist $f_1 \in E \cap Q$ and $f_2 \in (C\backslash E) \cap Q$ such that $\texttt{di}(f_1, f_2, \mathcal{I})$ does not hold. Since we have $E \cap Q = E\backslash(\bigcup_{H\in R} H)$ and $C \cap Q = C\backslash(\bigcup_{H\in R} H)$, we conclude that the set $\mathcal{I}$ is a complete test suite for $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$.

(c): We prove $\texttt{HA} \in \texttt{NP-complete}$. Given C, E, I, O, $\not\approx$, $\mathcal{I}$, $\mathcal{H}$, K as input of HA, checking whether some given $R \subseteq \mathcal{H}$ is such that $|R| \leq K$ and $\mathcal{I}$ is a complete test suite for $(C\backslash(\bigcup_{H\in R} H), E\backslash(\bigcup_{H\in R} H))$ requires polynomial time (we can prove it similarly as in the beginning of the proof of Theorem 2 (b)). Thus, $\texttt{HA} \in \texttt{NP}$. In order to prove $\texttt{HA} \in \texttt{NP-complete}$, we polynomially reduce an NP-complete problem, the *Set Cover* problem, to HA. The Set Cover problem, already considered in the proof of Theorem 1, is defined as follows: Given a set S, a collection J of subsets of S, and a natural number $K \in \mathbb{N}$, find a *set cover* J' for S (i.e., find a subset $J' \subseteq J$ such that every element in S belongs to at least one member of J') such that $|J'| \leq K$.

Given a problem instance of Set Cover, that is, a set $S = \{x_1, \ldots, x_n\}$, a collection of sets $J = \{S_1, \ldots, S_k\}$ with $S_j \subseteq S$ for all $1 \leq j \leq k$, and $K \in \mathbb{N}$, we will construct in polynomial time an HA instance from them, that is, we construct some C, E, I, O, $\not\approx$, $\mathcal{I}$, $\mathcal{H}$, K' from S, J, K, where C and E represent some computation formalism C and some specification E, respectively. This instance is such that the solution of HA for $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K')$ is *yes* iff the solution of Set Cover for (S, J, K) is *yes* as well. The idea of the transformation from a Set Cover instance to a HA instance is the following. We consider a test suite $\mathcal{I}$ with a single input i. Each element $x \in S$ is represented by a function in E and a function in $C\backslash E$, and there are no more functions in C. By defining responses of functions to input i in a similar way as in the proof of Theorem 2 (b), we make undistinguishable some specific pairs of functions. In particular, functions representing the *same* element $x \in S$ in E and $C\backslash E$ are made undistinguishable, and there are no more undistinguishable pairs of functions. In order to do this, each function produces some unique output α in response to i, and functions representing the same element $x \in S$ in E and $C\backslash E$ answer some shared output β too. This will make them undistinguishable, because we will consider $o_1 \not\approx o_2$ iff $o_1 \neq o_2$. For each set $\{x_1, \ldots, x_l\} \in J$, we have a hypothesis $H \in \mathcal{H}$ consisting of the functions representing the elements $x_1, \ldots, x_l$ in E, and there are no more hypotheses in $\mathcal{H}$. Solving HA requires that, for each pair of undistinguishable functions, at least one of them is removed by the set R of selected hypotheses. Only functions in E can be removed by hypotheses. Since all functions are involved in a pair of undistinguishable functions, solving HA consists in choosing some hypotheses such that *all* functions from E are removed. Since hypotheses represent sets in J, finding a set of hypotheses R such that all functions in E are removed and $|R| \leq K$ is equivalent to finding K or less sets of J such that they cover all elements in S. Thus, the answer of HA is yes iff the answer of Set Cover is yes.

Next we formally construct the HA instance $(C, E, I, O, \not\approx, \mathcal{I}, \mathcal{H}, K')$ from (S, J, K). Let us consider $S = \{x_1, \ldots, x_n\}$ and $J = \{S_1, \ldots, S_k\}$. Each element $x_j \in S$ is represented by a function $(outs^j) \in E$ and a function $(outs'^j) \in C\backslash E$, where $outs^j$ and $outs'^j$ denote the set of outputs answered by the corresponding functions when input i is given. The unique outputs α^j and α'^j are included in $outs^j$ and $outs'^j$, respectively. Besides, the output β^j is included in both $outs^j$ and $outs'^j$.

- $C = \{(outs^j) | 1 \leq j \leq n\} \bigcup \{(outs'^j) | 1 \leq j \leq n\}$ where:
  - for all $1 \leq j \leq n$ we have $outs^j = \{\alpha^j, \beta^j\}$
  - for all $1 \leq j \leq n$ we have $outs'^j = \{\alpha'^j, \beta^j\}$
- $E = \{(outs^j) | (outs^j) \in C\}$
- $I = \{i\}$
- $O = \{\alpha^j | 1 \leq j \leq n\} \cup \{\alpha'^j | 1 \leq j \leq n\} \cup \{\beta^j | 1 \leq j \leq n\}$
- $\not\approx = \{(o_1, o_2) | o_1, o_2 \in O \wedge o_1 \neq o_2\}$
- $\mathcal{I} = \{i\}$
- $\mathcal{H} = \{\{(outs^{j_1}), \ldots, (outs^{j_l})\} | \{x_{j_1}, \ldots, x_{j_l}\} \in J\}$
- $K' = K$

We consider the cost of constructing $(C, E, I, O, \not=, \mathcal{I}, \mathcal{H}, K')$ from (S, J, K). We have $|C| = 2 \cdot |S|$. For each function in C, the answer to a single input i is defined, and this answer consists of two outputs. Even if $\not=$ is extensionally defined, we have $|\not=| = |O|^2$, and we have $|O| = 3 \cdot |S|$. Besides, $|\mathcal{H}| = |J|$. Thus, $(C, E, I, O, \not=, \mathcal{I}, \mathcal{H}, K')$ can be constructed from (S, J, K) in polynomial time.

Let us show that the answer of HA to $(C, E, I, O, \not=, \mathcal{I}, \mathcal{H}, K')$ is *yes* iff the answer of the Minimum Set Cover to (S, J, K) is yes.

$\Longrightarrow$: Let us suppose that there exists $R \subseteq \mathcal{H}$ with $|R| \leq K'$ such that $\mathcal{I}$ is a complete test suite for $(C \backslash (\bigcup_{H \in R} H), E \backslash (\bigcup_{H \in R} H))$. By Lemma 3, we have that $f_1 \in \bigcup_{H \in R} H$ or $f_2 \in \bigcup_{H \in R} H$ for all $f_1 \in E$ and $f_2 \in C \backslash E$ such that $\mathtt{di}(f_1, f_2)$ does not hold. By the construction of $\mathcal{H}$, $H \subseteq E$ for all $H \in \mathcal{H}$. Besides, by the construction of C and E, for all function $f_1 \in E$ there exists $f_2 \in C \backslash E$ such that $\mathtt{di}(f_1, f_2, \mathcal{I})$ does not hold. We conclude $\bigcup_{H \in R} H = E$. Each function $f \in E$ represents an element $x \in S$ and, for all $H \in \mathcal{H}$, the set $H = \{(outs^{h_1}), \ldots, (outs^{h_j})\}$ represents a set $\{x_{h_1}, \ldots, x_{h_j}\} \in J$. Since $\bigcup_{H \in R} H = E$, R is a set cover of S. By construction, $K' = K$. Thus, we conclude $|R| \leq K$.

$\Longleftarrow$: Let $J' = \{S_1, \ldots, S_l\} \subseteq J$ be a set cover of S with $|J'| \leq K$. By the construction of $\mathcal{H}$, for all $H = \{(outs^{j_1}), \ldots, (outs^{j_l})\} \in \mathcal{H}$ there exists $S' \in J$ with $S' = \{x_{j_1}, \ldots, x_{j_l}\}$. Let $R \subseteq \mathcal{H}$ be such that, for all $S' \in J'$, we have $H \in R$ for some $H = \{(outs^{g_1}), \ldots, (outs^{g_k})\}$ with $S' = \{x_{g_1}, \ldots, x_{g_k}\}$, and there are no more hypotheses in R. Since J' is a set cover of S and all functions in E represent an element of S, we have $\bigcup_{H \in R} H = E$. By the definition of R from J' we have $|R| = |J'|$, so $|R| \leq K' = K$. Besides, $E \backslash (\bigcup_{H \in R} H) = \varnothing$ since $\bigcup_{H \in R} H = E$. Thus, $\mathcal{I}$ is a complete test suite for $(C \backslash (\bigcup_{H \in R} H), E \backslash (\bigcup_{H \in R} H))$.

□

7 Conclusions and Future Work

Formal Testing Techniques have reached a high level of maturity during the last few years. However, some common roots allowing us to relate testing methods with each other are still missing. In particular, a classification of testing problems would allow us to use our expertise about old testing problems to solve new problems. This paper tries to contribute to this (long term) goal. We have presented some general notions of testability and we have identified five classes of testability. We have studied some properties of the first class, i.e. *finitely testable* problems, including the complexity of finding a minimum complete test suite or measuring the completeness degree of incomplete test suites, alternative characterizations, transformations keeping the testability, the effect of adding testing hypotheses, and methods of reducing one testing problem into another. From a theoretical point of view, these techniques allow us to relate testing problems to each other as well as to classify them. From a practical point of view, they allow us to determine the (im-)possibility of finding complete test suites in different scenarios. They also allow us to reason about how far an incomplete test suite is from being complete, thus providing an implicit way to compare and select incomplete test suites. The proposed framework endows us with these capabilities even though it is highly abstract. In particular, many examples have been given to illustrate these capabilities, and new knowledge about some known testing frameworks and other useful settings have been discovered by applying our notions in the case studies section. Going one step further, the proposed general properties could be particularized and refined for specific computation formalisms.

More generally, properties presented in Section 4 allow us to envisage new paths to improve our understanding about testing in the future. This study should not be restricted to Class I; on the contrary, the properties of classes II, III, IV, and V must be studied in detail as well. In particular, the testing reduction notion could play a key role in organizing the five proposed classes into a big variety of subclasses that should be defined, related, and studied as well: e.g. *bounded* finite testability (what is the size of minimum complete test suites with respect to the size of the system representation? e.g. polynomial or exponential?), *adaptive* testability [23], *probabilistic* testability [32], *heuristic* testability, that is, using heuristics to find non-optimal (but good enough) test suites, etc.

Regarding adaptive testability, we will study the problem of testing systems in the case where the test suite is not set a priori, but the decision of which test is applied next can depend on results observed in response to previously applied tests. Let us consider the following example: We wish to check, by means of testing, whether some (terminating) programming function f from natural numbers to natural numbers fulfills the specification condition "$f(f(3)) = 4$". Since we do not know $f(3)$ a priori, any test suite constructed a priori must check *all* natural numbers, so this problem is not in Class I if only *preset* test suites are considered. However, if tests can depend on previous responses then we can apply input 3 and *next* input $f(3)$. Thus, the set $\{3, f(3)\}$ is a finite complete test suite in the adaptive testing case, and so this problem would belong to a kind of AdaptiveClass I. The differences between the classes presented in this paper (if they are considered as *preset*) and their corresponding adaptive counterparts should be studied. Also, we wish to study the applicability of the proposed testability notions to the related problem of *learnability* [16], [1].

Finally, we will develop a subdivision of Class II (the class of testing cases which are unboundedly-approachable by finite testing) in terms of the *speed of the convergence* towards distinguishing rate 1 (i.e. completeness) in the limit. In particular, given a pair (C, E) (and perhaps also a *specific* sequence $C_1 \subseteq C_2 \subseteq \ldots$ converging towards C), how does the size of finite test suites grows with respect to the required distinguishing

rate $\varepsilon < 1$? Would the size of required test suites be in e.g. $\mathcal{O}(n^2)$ with respect to $\frac{1}{1-\varepsilon}$? Or would it be in e.g. $\mathcal{O}(n \cdot log(n))$? These kind of asymptotic measures would allow us to know how fast we can converge, in each case, towards completeness in the limit. More importantly, this would provide us with a measure of *marginal utility* of tests: After applying n tests, what is the utility of applying one more test, in terms of gained distinguishing rate? When should we stop applying more tests, because the marginal utility of a new test (i.e. the gained distinguishing rate) would be too small? This would allow testers to calculate the appropriate amount of time they should assign to testing.

ACKNOWLEDGMENTS

We thank Fernando Rubio, Alberto de la Encina, Carlos Molinero, and Manuel Núñez for their suggestions, Diego Rodríguez for his support, and the anonymous reviewers of this paper as well as those of CONCUR'09 for their interesting comments.

REFERENCES

[1] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 351–369. ACM, 1992.

[2] G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specification: a theory and a tool. *Software Engineering Journal*, 6:387–405, 1991.

[3] I. Berrada, R. Castanet, P. Félix, and A. Salah. Test case minimization for real-time systems using timed bound traces. In *18th IFIP TC6/WG6.1 International Conference, TestCom 2006, LNCS 3964*, pages 289–305. Springer, 2006.

[4] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North Holland, 1976.

[5] E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 187–195. Springer, 2001.

[6] Y. Cheon and G.T. Leavens. A simple and practical approach to unit testing: The JML and JUnit way. In *16th European Conference on Object-oriented programming, ECOOP 2002, LNCS 2374*, pages 231–255. Springer, 2002.

[7] J.C. Cherniavsky and C.H. Smith. A recursion theoretic approach to program testing. *IEEE Transactions on Software Engineering*, 13:777–784, 1987.

[8] J.C. Cherniavsky and R. Statman. Testing: An abstract approach. In *Proceedings of the 2nd Workshop on Software Testing*, pages –. IEEE Computer Society Press, 1998.

[9] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.

[10] M. Davis and E. Weyuker. Metric space-based test-base adequacy criteria. *The Computer Journal*, 31(1):17–24, 1988.

[11] R.A. DeMillo, R.J. Lipton, and F.G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11:34–41, 1978.

[12] H. Do, G. Rothermel, and A. Kinneer. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis. *Empirical Software Engineering*, 11(1):33–70, 2006.

[13] R. Dorofeeva, K. El-Fakih, S. Maag, A.R. Cavalli, and N. Yevtushenko. FSM-based conformance testing methods: A survey annotated with experimental evaluation. *Information & Software Technology*, 52(12):1286–1297, 2010.

[14] C. Gaston, P. Le Gall, N. Rapin, and A. Touil. Symbolic execution techniques for test purpose definition. In *18th IFIP TC6/WG6.1 International Conference, TestCom 2006, LNCS 3964*, pages 1–18. Springer, 2006.

[15] M.-C. Gaudel. Testing can be formal, too. In *6th CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915*, pages 82–96. Springer, 1995.

[16] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[17] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[18] R.M. Hierons. Comparing test sets and criteria in the presence of test hypotheses and fault domains. *ACM Trans. on Software Engineering and Methodology*, 11(4):427–448, 2002.

[19] R.M. Hierons. Verdict functions in testing with a fault domain or test hypotheses. *ACM Transactions on Software Engineering and Methodology*, 18(4), 2009.

[20] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[21] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[22] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th Int. SPIN Workshop on Model Checking of Software, SPIN'04, LNCS 2989*, pages 109–126. Springer, 2004.

[23] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[24] N. López, M. Núñez, and I. Rodríguez. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science*, 353(1–3):228–248, 2006.

[25] K. Meinke. A stochastic theory of black-box software testing. In *Algebra, Meaning, and Computation*, volume 4060 of *LNCS*, pages 578–595. Springer, 2006.

[26] M. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–844, 2008.

[27] A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 196–205. Springer, 2001.

[28] I. Rodríguez. A general testability theory. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, LNCS 5710*, pages 572–586. Springer, 2009.

[29] I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

[30] K. Romanik and J.S. Vitter. Using vapnik-chervonenkis dimension to analyze the testing complexity of program segments. *Information and Computation*, 128(2):87–108, 1996.

[31] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

[32] M. Stoelinga and F. Vaandrager. A testing scenario for probabilistic automata. In *30th Int. Colloquium on Automata, Languages and Programming, ICALP'03, LNCS 2719*, pages 464–477. Springer, 2003.

[33] J. Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.

[34] J. Tretmans. Testing concurrent systems: A formal approach. In *10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.

[35] H. Zhu and X. He. A methodology of testing high-level petri nets. *Information and Software Technology*, 44(8):473 – 489, 2002.

Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2001 and his PhD in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. He also received the Best Paper Award in the IFIP WG 6.1 FORTE 2001 conference. Dr. Rodríguez has published more than 80 papers in international refereed conferences and journals. His research interests cover formal methods, testing techniques, swarm and evolutionary optimization methods, and functional programming.

Luis Llana is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Mathematics 1991 and his PhD in the same subject in 1996. His main research interest fields are formal methods and testing techniques. Currently he is opening new research fields such as artificial vision and e-learning.

Pablo Rabanal is an Assistant Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2004 and his PhD in the same subject in 2010, devoted to the development of nature-inspired techniques to solve NP-complete problems. Dr. Rabanal has published more than 20 papers in international refereed conferences and journals. His research interests cover swarm and evolutionary optimization methods, formal methods, testing techniques, and web services.