A General Testability Theory: Extended version *

Ismael Rodríguez

Dept. Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid, 28040 Madrid, Spain
e-mail: isrodrig@sip.ucm.es

Last update: February 23, 2011

Abstract. A general framework to reason about testing is developed. The difficulty of testing is assessed in terms of the amount of tests we must apply to precisely determine whether the system is correct or not. Based on this criterion, five testability classes are presented and related. Conditions enabling/disabling finite testability, as well as their relation with testing hypotheses, are explored. We measure how far incomplete test suites are from being complete, which allows to compare and select incomplete test suites. The complexity of finding minimum complete test suites is identified. We study how the problem of finding test suites to test a given kind of systems can be reduced to the problem of finding test suites for another kind of systems, which enables to export testing methods. Many typical examples from the formal testing techniques domain are used to illustrate how general notions are applied to specific cases.

Keywords: 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 given property or 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 [14, 17, 4, 19]. 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* [22, 7], *temporal systems* [20, 13, 2, 16], *probabilistic systems* [21, 15], *Java programs* [5, 6], 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

* This work is a revised and extended version of CONCUR'09 paper [18]. Work supported by projects TIN2006-15578-C02-01, TIN2009-14312-C02-01, and UCM-BSCH GR58/08 - group number 910606.

applying them to the implementation, the results allow to precisely determine whether the implementation is correct or not with respect to the specification. Let us note that constructing and applying complete test suites is often unfeasible. 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 [10, 19, 11]. By assuming hypotheses, the number of systems that could actually *be* the implementation is reduced (provided that assumed hypotheses actually hold), so less tests must be applied to seek for undesirable behaviors. Actually, this may allow to reduce the number of necessary tests from infinite to finite [1, 8]. However, the specific conditions that make the *difference* between requiring infinite or finite sets of tests (regardless of the use of hypotheses) must be studied. In this paper, by *testability* we will understand the difficulty 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 before (that is, complete test suites are (a) finite; (b) infinite, but any arbitrarily high degree of partial completeness can be finitely achieved; (c) countable infinite, but not as in (b); (d) uncountable infinite; or (e) there does not exist any complete test suite at all) must be analyzed. To the best of our knowledge, the previous five general testability scenarios have never been defined or investigated. In particular, the border between them has never been studied as a general issue, that is, by means of a framework where any kind of formalism for defining specifications and implementations fits (only some particular hypotheses enabling finite complete test suites for some particular models have been reported). This contrasts with other mature fields of Computer Science like Computability or Complexity, where a well established theory allows to relate different known problems with 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* because techniques are not easily inherited from a problem to another – even if they are quite similar after abstracting factors not directly affecting testability.

We propose a first step towards the construction of a general testability theory. We present some formal criteria to classify testing problems according to their testability. Providing a complete testability hierarchy is a huge task and is out of the goals of the paper. Instead, a hierarchy including only five main classes of testability (*finitely testable*, *unboundedly-approachable finitely testable*, *infinite countable testable*, *infinite uncountable testable*, and *untestable*) is presented. Some formal properties of the *finitely testable* class are presented, 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 apply these properties to study the testability of some examples, both well-known and ad hoc. A deeper study of the properties of the remaining four testability classes is left as future work.

The contribution of this paper is twofold. From a theoretical point of view, the proposed techniques allow to classify and relate testing problems with each other, thus making a first step towards constructing a complete testability hierarchy and improving our understanding about testing. From a practical point of view, being able to reason about the most *ideal* testing scenario (that is, the case where completeness is feasible) allows to deal with more practical scenarios. Though test suites are incomplete in most of practical cases, identifying the additional conditions required by an incomplete test suite to *become* complete allows to compare and select good (incomplete) test suites: We prefer those test suites such that the conditions required by them to be complete are *weaker* or *more feasible*. Besides, if testing problems can be related then strategies allowing to find good test suites for a given kind of systems can be exported to find good test suites in other testing scenarios. Moreover, testability issues could affect the design of systems: If there are several suitable alternatives for a design and, for one of them, the problem of testing the corresponding system would belong to a better class (or incomplete test suites would be *closer* to be complete), then this alternative is preferable.

This paper is structured as follows. In the next section we construct some basic concepts to reason about testability and we present the classes of our testability hierarchy. In Section 3 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. Proofs are given in the appendix of the paper.

2 Testability concepts

In this section we introduce some preliminary concepts and we define testability classes. First, we present a general notion to denote implementations and specifications in our framework. Since testing consists in studying systems in terms of their observable behavior, the behavior of a system can be defined by a function

relating inputs with their possible outputs. Next we assume that 2^S denotes the powerset of the set S.

Definition 1. Let I be a set of *input symbols* and O be a set of *output symbols*. 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 $f(i) \neq \emptyset$. □

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. We illustrate it in the next example.

Example 1. 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 1 (respectively, the set O) is the set of all *sequences* of symbols belonging to I' (resp. to O'), that is, $I = I'^*$ and $O = O'^*$.[1] 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 C represent the set of all programs in a Turing-complete language, say Java. We codify the interaction of a user with a program by using an appropriate notation. For example, the behavior of a given program P could be denoted by a function f such that, in particular, we have

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

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

[1] When dealing with FSMs we will assume the same meaning for I', O', I, and O.

will be 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 Java 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 one after each other 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.[2] Alternatively, we may choose not to assume this independence. In this case, an interaction where we apply i_1 and next i_2 is *not* assumed to return o_1 and next o_2, for some $o_1 \in f(i_1)$ and $o_2 \in f(i_2)$. Instead, it returns o for some $o \in f(i')$, where i' is a *single* input representing an interaction where we apply i_1 and next we apply i_2. Hence, scenarios where the independence of input applications is not guaranteed can also be represented in the proposed formalism.

Definition 2. Let C be a computation formalism. A *specification* of C is a set $E \subseteq C$. □

[2] 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.

If $f \in E$ then f denotes a *correct* behavior, while $f \in C \backslash E$ denotes that f is incorrect. Thus, a specification implicitly denotes a correctness criterion. For example, let us consider two functions $f, f' \in C$ be 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 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 B is correct with respect to A. Then we could define E as follows: $E = \{f' | f \preceq f' \wedge f' \in C\}$.

Testing provides verdicts in terms of the outputs collected by interacting with the IUT. It is worth to point out that, in some situations, there might exist two different outputs that are not *distinguishable* in practice by means of observation. This is specially relevant if one of them is produced when the machine fulfills the specification and the other one is given when the machine is incorrect, because then the observed output does not allow to assess the correctness of the observed IUT.

Definition 3. Let O be a set of outputs. A *distinguishing relation* for O is an anti-reflexive symmetric binary relation $\mathcal{D}$ over O. We denote the complementary of $\mathcal{D}$ by $\not\mathcal{D}$. □

Example 2. We revisit the example where the computation formalism C represents the set of all FSMs. If after receiving some $i \in I$ the IUT can answer either $o_1 \in O$ or $o_2 \in O$ then, by observing the actual response of the IUT, we can decide which of these sequences is produced. Hence, we may consider a trivial distinguishing relation $\mathcal{D}$ where two outputs $o_1, o_2 \in O$ are distinguishable iff they represent different sequences of O' outputs, i.e. $o_1 \mathcal{D} o_2$ iff $o_1 \neq o_2$.

This strong distinguishing capability may not be feasible in other frameworks. We revisit the example where C represents Java programs. Let us suppose that the time at which messages are produced is *not* represented, that is, outputs only represent sequences of messages. Let us consider outputs $o_1 = \langle mes_1, mes_2, stop \rangle$ and $o_2 = \langle mes_1, \bot \rangle$. 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 $o_1 \mathcal{D} o_2$ iff (i) $o_1 \neq o_2$; *and* (ii) neither $o_1 = w \cdot \bot$ nor $o_2 = w \cdot \bot$, where w is the longest common prefix of o_1 and o_2. Given this definition of $\mathcal{D}$, if we observe $o \in \{o_1, o_2\}$ and we have $o_1 \mathcal{D} o_2$ then we can decide whether $o = o_1$ or $o = o_2$ in finite time indeed.[3]

It is worth to point out that, if outputs are given as in Example 1 (that is, a time stamp is attached to each O' output), then non-termination issues do not affect the distinguishability of outputs. Let us consider $o = \langle (t_1, o_1), \ldots, (t_n, o_n) \rangle \in$

[3] Alternatively, a tester could assume a kind of *non-termination observability*, which is common 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 $o_1 \mathcal{D} o_2$ iff $o_1 \neq o_2$.

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 $o_1 \mathcal{D} o_2$ iff $o_1 \neq o_2$. □

Next we identify *complete test suites*, i.e. sets of inputs such that, if they are applied to the IUT, then collected outputs allow to precisely determine if the IUT fulfills the considered specification or not. More precisely, a complete test suite is a set of inputs such that, for all correct function f and incorrect function f', there is at least one input i in the set such that the sets of outputs that can be produced by f and f' in response to i are *pairwise distinguishable* (i.e. all outputs in one set are distinguishable from all outputs belonging to the other set).

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

Let $f \in C$. We denote by $\texttt{pairs}\,(f, \mathcal{I})$ the set of all pairs $(i, f(i))$ such that $i \in \mathcal{I}$.

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 exist $i \in \mathcal{I}$, $(i, outs) \in \texttt{pairs}\,(f, \mathcal{I})$, and $(i, outs') \in \texttt{pairs}\,(f', \mathcal{I})$ such that for all $o \in outs$ and $o' \in outs'$ we have $o \mathcal{D} o'$.

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

2.1 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 in Section 2.2).

Definition 5. Let C be a computation formalism for I and O, $E \subseteq C$ be a specification, and $\mathcal{D}$ be a distinguishing relation.

We say that a triple $(C, E, \mathcal{D})$ is *finitely testable* if there exists a finite complete test suite for C, E, and $\mathcal{D}$. We denote the set of all finitely testable triples $(C, E, \mathcal{D})$ by `Class I`.

We say that $(C, E, \mathcal{D})$ is *countable testable* if there exists a countable complete test suite for C, E, and $\mathcal{D}$. We denote the set of all countable testable triples $(C, E, \mathcal{D})$ by `Class III`.

We say that $(C, E, \mathcal{D})$ is *testable* if there exists a complete test suite for C, E, and $\mathcal{D}$. We denote the set of all testable triples $(C, E, \mathcal{D})$ by `Class IV`.

We denote the set of all triples $(C, E, \mathcal{D})$ by `Class V`. □

This classification induces the relation `Class I` $\subseteq$ `Class III` $\subseteq$ `Class IV` $\subseteq$ `Class V`. 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 example given for a triple in `Class IV` but not in `Class III` will be used to argue that the interest of `Class IV` is, probably, 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 $E_1 \subseteq C_1$, $\mathcal{D}$ be the trivial distinguishing relation (i.e., $o_1 \mathcal{D} o_2$ iff $o_1 \neq o_2$), 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 $\mathcal{I}_1$ [14]. 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, $(C_1, E_1, \mathcal{D}) \in$ `Class I`. As we will see later in Example 5 (given in Section 3) this result can also be proved by applying forthcoming Lemma 2 (c) (this lemma, given right before Example 5, 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 $(C_2, E_2, \mathcal{D}) \notin$ `Class I` (in Section 3 we also prove this property in Example 5, this time by using forthcoming Lemma 2 (b)). However, the set of all sequences of I' inputs, that is I'^*, distinguishes all pairs of non-equivalent deterministic FSMs. In fact, I'^* can be numbered (e.g., lexicographically). Thus, we have $(C_2, E_2, \mathcal{D}) \in$ `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 $(C_3, E_3, \mathcal{D})$, so $(C_3, E_3, \mathcal{D}) \in$ `Class IV`. However, in general we have $(C_3, E_3, \mathcal{D}) \notin$ `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, $\mathcal{D}$: 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 input sequences is so, the set of all sequences of rational-timed inputs is countable as well (a proof of this can be found in the appendix). Hence we have $(C'_3, E'_3, \mathcal{D}) \in$ `Class III`. 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 triples in `Class IV` but not in `Class III`, these triples lie in a similar idea as $(C_3, E_3, \mathcal{D})$. Despite of the doubtable practical interest of `Class IV` (in terms of kind of new triples it adds to `Class III`), 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 with the same behavior as M_1 for all inputs but 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 of 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 $(C_4, E_4, \mathcal{D}) \in$ `Class V` but $(C_4, E_4, \mathcal{D}) \notin$ `Class IV`.[4]

Non-determinism does *not* imply that finite testability is not possible. Let $C_5 = \{f, f'\}$ 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, E_5, and $\mathcal{D}$. Thus, $(C_5, E_5, \mathcal{D}) \in$ `Class I`.

Finally let us note that, given C, E, and $\mathcal{D}$, $(C, E, \mathcal{D}) \in$ `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 Java-written functions from integers to integers 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 FSMs cannot express all functions in C or all functions in E, $\{3, 5\}$ is a complete test suite for $(C, E, \mathcal{D})$: For all $f \in C$, $f(3) < f(5)$ iff $f \in E$. So, $(C, E, \mathcal{D}) \in$ `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. □

Let $\mathcal{D}$ be defined in such a way that $o_1 \ \mathcal{D} \ o_2$ only if o_1 and o_2 can be effectively distinguished via observation (see Example 2). If $(C, E, \mathcal{D}) \in$ `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 (this idea will be further elaborated later in Definition 6). 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

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

order is not enough to reach complete coverage because the complete test suite is not countable.

The next result shows the expectable property that observations collected by complete test suites univocally determine whether the IUT is correct. We use the term $\mathcal{G}$ to denote any set of pairs of inputs and outputs (i, o) we could collect by testing a given system. For instance, let $f(i_1) = \{o_1, o_2\}$ and $f(i_2) = \{o_3, o_4\}$. If we apply a test suite $\mathcal{I} = \{i_1, i_2\}$ to an IUT behaving like f, then we could obtain e.g. $\mathcal{G} = \{(i_1, o_2), (i_2, o_3)\}$.

Lemma 1. Let C be a computation formalism for I and O, $E \subseteq C$ be a specification, and $\mathcal{D}$ be a distinguishing relation. Let $\mathcal{I}$ be a complete test suite for C, E, and $\mathcal{D}$. Let $f \in C$ and $\mathcal{G} \in 2^{\mathcal{I} \times O}$ be a set of pairs of inputs and outputs such that for all $(i, o) \in \mathcal{G}$ we have $o \in f(i)$.

(1) If $f \in E$ then there does not exist $f' \in C \backslash E$ such that for all $i \in \mathcal{I}$ there exist $o' \in f'(i)$ and $(i, o) \in \mathcal{G}$ with $o \not\mathcal{D} o'$.
(2) If $f \in C \backslash E$ then there does not exist $f' \in E$ such that for all $i \in \mathcal{I}$ there exist $o' \in f'(i)$ and $(i, o) \in \mathcal{G}$ with $o \not\mathcal{D} o'$. □

Before presenting `Class II` in the next section, we discuss the relation between complete test suites, as presented in Definition 4, 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. By Lemma 1, each correct function is distinguished from each incorrect function for some $i \in \mathcal{I}$ (and viceversa). Let us suppose that the distinguishing relation $\mathcal{D}$ 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 2, 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 $\mathcal{D}$, 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.[5] 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 $\mathcal{I}$ are applied in *parallel* or *interleaving*: Since each required distinction is provided in finite time, once all distinctions are reached we can stop the interleaving/parallel 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 3 also apply to this alternative notion of complete test suite, once a straightforward adaptation is made in each case.

[5] This is the case if the specification *permits* not to terminate for some inputs.

2.2 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. In some cases where finite testability is not possible, it may still be possible to test the IUT up to any arbitrarily high *confidence degree* with a *finite* test suite. By confidence degree, here we mean 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. If, for all real value $0 \leq \varepsilon < 1$, we can find a *finite* test suite providing a ratio higher than ε, then we say that the IUT is unboundedly-approachable by finite testing. Care must be taken to measure this ratio when (countable) *infinite* computation formalisms are considered. For instance, even if a test suite distinguishes one out of two pairs of correct/incorrect functions, and thus we expect to reach a 0.5 ratio, the set of all pairs and the set of distinguished pairs have the same cardinality (the same as $\mathbb{N}$). Instead of considering the cardinality of these infinite sets, the ratio will be measured for some *finite* subsets of the computation formalism. These subsets will be defined in such a way that they tend to cover the whole computation formalism as they increase.

Definition 6. Let C be a countable computation formalism for I and O, $E \subseteq C$ be a specification, $\mathcal{D}$ be a distinguishing relation, and $\mathcal{I} \subseteq I$ be a set of inputs. Let $h : \mathbb{N} \longrightarrow C$ be a bijective function and $k \in \mathbb{N}$. The *restriction* of C to k in h, denoted by $C \downarrow_k h$, is defined as $\{h(j) | 0 \leq j \leq k\}$. The *restriction* of E to k in h, denoted by $E \downarrow_k h$, is defined as $(C \downarrow_k h) \cap E$.

If C is finite then we define the *distinguishing rate* of $\mathcal{I}$ for $(C, E, \mathcal{D})$, denoted by $\texttt{d-rate}\,(\mathcal{I}, C, E, \mathcal{D})$, as $\dfrac{|\{(f, f') | f \in E, f' \in C \backslash E, \texttt{di}\,(f, f', \mathcal{I})\}|}{|E| \cdot |C \backslash E|}$.

We say that $(C, E, \mathcal{D})$ is *unboundedly-approachable by finite testing through* h if for all $\varepsilon \in \mathbb{R}$, with $0 \leq \varepsilon < 1$, there exists a finite set of inputs $\mathcal{I} \subseteq I$ such that, for all $k \in \mathbb{N}$, we have $\texttt{d-rate}\,(\mathcal{I}, C \downarrow_{k_1} h, E \downarrow_{k_1} h, \mathcal{D}) \geq \varepsilon$ for some $k_1 \geq k$. □

Clearly, the property of being unboundedly-approachable by finite testing strongly depends on the function h considered to sort the computation formalism. In particular, given a triple $(C, E, \mathcal{D})$, it may be unboundedly-approachable for some sorting functions but not for others. Still, we can define a class of unboundedly-approachable triples $(C, E, \mathcal{D})$ as follows: We say that $(C, E, \mathcal{D})$ is *unboundedly-approachable* if there exists h such that $(C, E, \mathcal{D})$ is unboundedly-approachable by finite testing through h, and we denote by `Class II` the set of all triples $(C, E, \mathcal{D})$ that are either unboundedly-approachable or finitely testable.[6] This class is related with the other classes as expected: We have `Class I` $\subseteq$ `Class II` $\subseteq$ `Class III`. In fact, as the next example shows, both inclusions are proper.

[6] We *explicitly* include finitely testable triples in `Class II` because, if $(C, E, \mathcal{D}) \in$ `Class I` and C is *finite*, then any distinguishing rate can be reached for $(C, E, \mathcal{D})$ with a finite test suite (in fact, even 1), but there is no h such that $(C, E, \mathcal{D})$ is unboundedly approachable through h (note that we require that h is a *bijection* from $\mathbb{N}$ to C).

Example 4. We revisit $(C_2, E_2, \mathcal{D})$ 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.

We have $(C_2, E_2, \mathcal{D}) \in$ `Class II`, i.e. $(C_2, E_2, \mathcal{D})$ is unboundedly-approachable by finite testing through some bijective function $h : \mathbb{N} \longrightarrow C_2$. The proof of this result is presented in the appendix, next we present an intuition. For all $l \in \mathbb{N}$, let $\mathcal{I}^l$ denote the set of all input sequences of length l. We define a function h sorting C_2 in such a way that, for all $l \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that all possible ways to react to $\mathcal{I}^l$ appear in the *same* proportion in $C_2 \downarrow_k h$ (i.e., the number of functions providing each combination of responses to $\mathcal{I}^l$ is the same for all combinations). Only one of these ways to react to $\mathcal{I}^l$ conforms to f_1. Thus, if there are n ways to react to $\mathcal{I}^l$, then $\mathcal{I}^l$ reaches a distinguishing rate $1 - \frac{1}{n}$. In longer restrictions $C_2 \downarrow_{k_1} h$ with $k_1 > k$, the proportion of functions providing each possible response to $\mathcal{I}^l$ is preserved, so $\mathcal{I}^l$ also reaches a $1 - \frac{1}{n}$ rate in these restrictions. Let $0 \leq \varepsilon < 1$. In order to find a finite test suite providing a rate higher than ε for *all* restrictions, we just have to consider $l' \in \mathbb{N}$ such that there are m ways to react to $\mathcal{I}^{l'}$, with $\frac{1}{m} < 1 - \varepsilon$. For details, see the proof in the appendix.

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. There does not exist any bijective function $h : \mathbb{N} \longrightarrow C_2'$ such that $(C_2', E_2, \mathcal{D})$ is unboundedly-approachable by finite testing through h. 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 $|\mathcal{I}|$). Thus, for all $\mathcal{I}$, h, and $0 < \varepsilon < 1$ there exists $k \in \mathbb{N}$ such that, for all $k_1 > k$, we have `d-rate`$(\mathcal{I}, C_2' \downarrow_{k_1} h, E_2 \downarrow_{k_1} h, \mathcal{D}) < \varepsilon$. We conclude $(C_2', E_2, \mathcal{D}) \notin$ `Class II`. □

The two cases considered in the previous example illustrate an interesting fact: 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.

3 Studying properties of Class I

In this section we study the properties of Class I. Conditions required for finite testability are shown, and some alternative characterizations are presented. We define transformations keeping the testability, and we analyze the effect of adding testing hypotheses. The complexity of finding minimum complete test suites and measuring how far a test suite is from being complete is analyzed. We also 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 out of the scope of this paper and 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 outputs O, $E \subseteq C$ is a specification, and $\mathcal{D}$ is a distinguishing relation.

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

Lemma 2. 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 $o_1 \not\mathcal{D} o_2$ for some $o_1 \in f(i)$, $o_2 \in f'(i)$. Then, $(C, E, \mathcal{D}) \notin$ `Class IV`.

(b) $(C, E, \mathcal{D}) \notin$ `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 $o_1 \not\mathcal{D} o_2$ 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 $o_1 \not\mathcal{D} o_2$ for some $o_1 \in f(i)$, $o_2 \in f(i)$. Then, $(C, E, \mathcal{D}) \in$ `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 $f(i) \neq f'(i)$ and, for all $i' \neq i$ with $i' \in I$, we have $f(i') = f'(i')$. Then, $(C, E, \mathcal{D}) \notin$ `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. We illustrate this in the following example.

Example 5. Let $(C_2, E_2, \mathcal{D})$ be defined as in Example 3. We argued before that, in general, $(C_2, E_2, \mathcal{D}) \notin$ `Class I`. Now we prove it by applying Lemma 2 (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 2 (b), $(C_2, E_2, \mathcal{D}) \notin$ `Class I`.

Let $(C_1, E_1, \mathcal{D})$ be defined as in Example 3. We provide a simple proof of $(C_1, E_1, \mathcal{D}) \in$ `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 2 (c) we conclude $(C_1, E_1, \mathcal{D}) \in$ `Class I`. Let us apply Lemma 2 (c) in a similar way, but in a very different context. Let P denote the set of all deterministic programs written in some Turing-complete language (say Java) with less than k_1 characters, I denote the (finite) set of *keys* in a standard keyboard, and O denote the (finite) set of *ASCII symbols*, 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 k_2 ticks then we consider $f(i) = \{Null\}$ with $Null \in O$. Let $E_1' = \{f\}$ for some $f \in C_1'$ and $\mathcal{D}$ be the trivial distinguishing relation. Since C_1' is finite and all functions in C_1' are distinguishable, again by Lemma 2 (c) we have $(C_1', E_1', \mathcal{D}) \in$ `Class I`.

Lemma 2 (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 triple $(C, E, \mathcal{D})$ considered in the last paragraph of Example 3 in Section 2.1, dealing with an *infinite* number of Java 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 such that, if they receive $a \in I'$ 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, and the trivial distinguishing relation $\mathcal{D}$, 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, E_6 and the relation $\mathcal{D}$, and we have $(C_6, E_6, \mathcal{D}) \in$ `Class I`.

Let us consider again programs written in a Turing-complete language, say Java again. Let P denote the set of all *terminating* deterministic programs written in Java 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 Java program p and a sequence

of characters i, we can always construct another deterministic Java program p' such that p' behaves as p for all sequences of characters but i.[7] 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 2 (d) we conclude $(C_1'', E_1'', \mathcal{D}) \notin$ `Class I`.

Let us revisit $(C_6, E_6, \mathcal{D})$ given before in this example and let $f \in E_6$. We cannot find, for all $i \in I$, a function $f' \in C_6\backslash E_6$ such that f and f' only differ for 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 2 (d) to infer $(C_6, E_6, \mathcal{D}) \notin$ `Class I`. □

The next results analyze conditions preserving testability, that is, we study how we can modify C or E in such a way that complete test suites are preserved. In the next set of properties, the symmetric property of (c) also holds (i.e. we can flip the roles of correct/incorrect functions in (c) in such a way that an *incorrect* function f is removed).

Lemma 3. *(Equi-testability Lemma)* Let Q denote the set of all tuples (f, f', B) where $f \in E$, $f' \in C\backslash E$, and B is the set of all inputs $i \in I$ such that for all $o \in f(i)$, $o' \in f'(i)$ we have $o \,\mathcal{D}\, o'$. We have the following properties:

(a) $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$ iff for all $(f, f', B) \in Q$ we have $B \cap \mathcal{I} \neq \emptyset$.
(b) Let $(f_1, f_1', B_1), (f_2, f_2', B_2) \in Q$ be such that $B_1 \subseteq B_2$ and $B_1 \neq \emptyset$. Let $Q' = Q\backslash\{(f_2, f_2', B_2)\}$. Then, $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$ iff for all $(f, f', B) \in Q'$ we have $B \cap \mathcal{I} \neq \emptyset$.
(c) Let $f \in E$ and $J = \{B | f' \in C\backslash E \wedge (f, f', B) \in Q\}$ (that is, J consists of all sets of inputs distinguishing f from each incorrect function). Let us suppose that for all $B \in J$ there exists $(g, g', B') \in Q$ with $g \neq f$ such that $B' \subseteq B$ and $B' \neq \emptyset$. Then, $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$ iff $\mathcal{I}$ is a complete test suite for $C\backslash\{f\}$, $E\backslash\{f\}$, and $\mathcal{D}$. □

Example 6. We apply Lemma 3 (c) to dramatically reduce the number of pairs of functions we must consider to achieve complete testing in a specific case. Let us revisit C_6 and E_6 as they are defined in Example 5. For any function $f \in E_6$, f can be distinguished from all functions $f' \in C_6\backslash E_6$ by input a. Let us remove from C_6 and E_6 all correct functions but one, and let C_6' and E_6' be the resulting sets. By Lemma 3 (c), a complete test suite for C_6, E_6, and $\mathcal{D}$ is a complete test suite for C_6', E_6', and $\mathcal{D}$, and viceversa. Similarly, we remove from C_6' all incorrect functions but one, leading to set C_6''. In this way, the problem of testing the infinite set E_6 against the infinite set C_6 is reduced to the problem of testing $E_6' = \{f\}$ against $C_6'' = \{f, f'\}$ for some functions f, f' with $f \in E_6$ and $f' \in C_6\backslash E_6$. Since C_6 is finite and we can distinguish f from f' (by applying input a), by Lemma 2 (c) we have $(C_6'', E_6', \mathcal{D}) \in$ `Class I`. Thus, by Lemma 3 (c) we rediscover $(C_6, E_6, \mathcal{D}) \in$ `Class I`. □

[7] The program p' can be trivially built as follows: p' consists of an *if-then-else* statement where we check if the input sequence is i; if it is then we provide an answer different to the one given by p else the do as p.

A very different approach to reduce a testing problem into another will be presented later in Definition 10.

Next, we provide a result allowing to find 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 set of *sets of inputs* such that, for all set of inputs $A \in \mathcal{A}$, we have that

(1) for all $A' \in \mathcal{A}$ with $A' \neq A$ we have $A' \cap A = \emptyset$, and
(2) there exist $f \in E$, $f' \in C \backslash E$ with $\{i \mid i \in I \wedge \forall\, o \in f(i), o' \in f'(i) : o \,\mathcal{D}\, o'\} \subseteq A$.

If $\mathcal{A}$ is finite then either $(C, E, \mathcal{D}) \notin$ `Class I` or for all finite complete test suite $\mathcal{I}$ for C, E, and $\mathcal{D}$ we have $|\mathcal{I}| \geq |\mathcal{A}|$. If $\mathcal{A}$ is infinite then $(C, E, \mathcal{D}) \notin$ `Class I`. □

Example 7. Let C_2, E_2, and $\mathcal{D}$ be defined as in Example 3. In particular, let $E_2 = \{f\}$ consist of a single correct function, $I = I'^*$ denote 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 $Q_{k_1} \cap Q_{k_2} = \emptyset$. Thus, the infinite set $\mathcal{A} = \{Q_k | k \in \mathbb{N}\}$ fulfills conditions (1) and (2) of Proposition 1. Hence, we rediscover $(C_2, E_2, \mathcal{D}) \notin$ `Class I`.

It is worth to point out that we may have $(C, E, \mathcal{D}) \notin$ `Class I` even if there does not exist any infinite set $\mathcal{A}$ fulfilling conditions (1) and (2) of Proposition 1 (that is, the longest set fulfilling (1) and (2) is finite). This is proved in the appendix by explicitly constructing a case where this happens. □

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,

$$G = \left\{ \left\{ i \,\middle|\, \begin{array}{l} i \in I \wedge \\ \forall\, o \in f(i), o' \in f'(i) : o \,\mathcal{D}\, o' \end{array} \right\} \,\middle|\, \begin{array}{l} f \in E, \\ f' \in C \backslash E \end{array} \right\}$$

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

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

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

(c) for each function $f \in C$, a tuple $(f(i_1), \ldots, f(i_k))$ denoting the possible outputs of f for all inputs is explicitly provided.

We will denote the problem of finding the minimum complete test suite in this case as the *Minimum Complete Suite* problem. Next we prove its NP-completeness (in particular, a polynomial reduction from the *Minimum Set Cover* problem to this problem is constructed in the appendix).

Definition 7. Let C be a finite computation formalism for the finite sets of inputs and outputs $I=\{i_1, \ldots, i_k\}$ and O, respectively, $E \subseteq C$ be a finite specification, and $\mathcal{D}$ be a finite distinguishing relation. Let C' and $E' \subseteq C'$ be sets of tuples representing the behavior of functions of C and E respectively; formally, for all $f \in C$ we have $(f(i_1), \ldots, f(i_k)) \in C'$ and viceversa, and for all $g \in E$ we have $(g(i_1), \ldots, g(i_k)) \in E'$ and viceversa.

Given C', E', I, O, $\mathcal{D}$, and some $K \in \mathbb{N}$, the *Minimum Complete Suite* problem (`MCS`) is defined as follows: Is there any complete test suite $\mathcal{I}$ for C, E, and $\mathcal{D}$ such that $|\mathcal{I}| \leq K$? □

Theorem 1. `MCS` $\in$ `NP-complete`. □

3.1 Dealing with testing hypotheses

In this section we present some notions allowing 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 an hypothesis by the set H of systems we *remove* from those that could actually be the IUT if the hypothesis is assumed.

Definition 8. 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, E, $\mathcal{D}$. If $\mathcal{I}$ is a complete test suite for $C \backslash H$, $E \backslash H$, $\mathcal{D}$ then we say that H *enables* $\mathcal{I}$ for $(C, E, \mathcal{D})$.

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

Lemma 4. Let $\mathcal{I} \subseteq I$ be a finite set of inputs that is *not* a complete test suite for C, E, and $\mathcal{D}$. Let H be a testing hypothesis for C. We have that H *enables* $\mathcal{I}$ for $(C, E, \mathcal{D})$ iff for all $f \in E$, $f' \in C \backslash E$ such that for all $i \in \mathcal{I}$ there exist $o \in f(i)$, $o' \in f'(i)$ with $o \not\mathcal{D} o'$, we have $f \in H$ or $f' \in H$. □

In Section 2.2 we proposed to use the *distinguishing rate* (see Definition 6) 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 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. 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 `MCS` problem in Definition 7. That is, C, E, I, O are finite and functions in C and E are explicitly denoted by some sets of tuples C' and E'. 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 9. Let us consider the same preamble as in Definition 7, that is: Let C be a finite computation formalism for the finite sets of inputs and outputs $I = \{i_1, \ldots, i_k\}$ and O, respectively, $E \subseteq C$ be a finite specification, and $\mathcal{D}$ be a finite distinguishing relation. Let C' and $E' \subseteq C'$ be sets of tuples representing the behavior of functions of C and E respectively; formally, for all $f \in C$ we have $(f(i_1), \ldots, f(i_k)) \in C'$ and viceversa, and for all $g \in E$ we have $(g(i_1), \ldots, g(i_k)) \in E'$ and viceversa.

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, $\mathcal{D}$, $\mathcal{I}$ and some $K \in \mathbb{N}$, the *Minimum Function Removal* problem (`MFR`) 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 $(C \backslash R, E \backslash R, \mathcal{D})$?

Given C', E', I, O, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{H}$, and some $K \in \mathbb{N}$, the *Minimum Function removal via Hypotheses* problem (`MFH`) is defined as follows: Is there any set of hypotheses $R \subseteq \mathcal{H}$ with $|\bigcup_{H \in \mathcal{H}} H| \leq K$ such that $\mathcal{I}$ is a complete test suite for $(C \backslash (\bigcup_{H \in R} H), E \backslash (\bigcup_{H \in R} H), \mathcal{D})$?

Given C', E', I, O, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{H}$, and some $K \in \mathbb{N}$, the *Minimum Hypotheses Assumption* problem (`MHA`) 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 $(C \backslash (\bigcup_{H \in R} H), E \backslash (\bigcup_{H \in R} H), \mathcal{D})$? $\square$

The differences among the previous problems (and the corresponding coverage measures they allow to calculate) is the following: In MFR, the coverage is measured in terms of the minimal *number* of functions that must be removed, where *any* subset of C is allowed to be removed. In MFH, 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, MHA considers the coverage in terms of the number of assumed *hypotheses*, rather than on the number of removed functions. At a first glance, one might think that all MFR, MFH, and MHA are NP-hard problems. Interestingly, MFR is not, 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 [12]. 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 introduces the proposed reduction and uses it to actually find the *minimum* function removal in time $\mathcal{O}(|C'|^{5/2} + |C'|^2 \cdot |\mathcal{I}| \cdot |\mathcal{O}|^2)$, thus solving MFR polynomially. On the other hand, the NP-completeness of MFH and MHA is proved by constructing polynomial reductions from *3-SAT* and *Minimum Set Cover*, respectively.

Theorem 2. We have the following properties:
(a) MFR $\in$ P
(b) MFH $\in$ NP-complete
(c) MHA $\in$ NP-complete □

3.2 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 fix computation formalism for all possible specifications, a (possibly infinite) set of triples $(C, E, \mathcal{D})$ 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 triples $(C, E, \mathcal{D})$, and the *goal* of a testing problem will be, given any triple in the set, finding a complete test suite for that triple. 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.

Definition 10. A *testing problem* $\mathcal{T}$ for a set of inputs I is a set $\mathcal{T}$ of triples $(C, E, \mathcal{D})$ with the usual meanings of $C, E, \mathcal{D}$ where, for all $(C, E, \mathcal{D}) \in \mathcal{T}$, 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 $f : \mathcal{T} \longrightarrow 2^I$ is a *finite suite derivation* for $\mathcal{T}$ if, for all $p \in \mathcal{T}$, $f(p)$ 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`. □

We should not confuse $\leq_F$ with the classical *testing preorder* relation given in [9]. In particular, the $\leq_F$ relation does not compare processes but *testing problems*. In this sense, it reminds the reductions of computability and complexity theory, where a computation problem is transformed into another problem.[8] 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.

Example 8. We illustrate $\leq_F$ with examples. These examples will lie in manipulating *finite* computation formalisms and *mapping* computation formalisms into each other. Other kinds of examples, involving infinite computation formalisms that cannot be transformed into each other, will be considered later in Example 9.

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 the 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.[9] 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 $\mathcal{T}_{FSM} =$

[8] In fact, a testing problem also represents a kind of computability problem.
[9] TEFSMs can be thought as a kind of *timed automata*.

$\{(C^{FSM}_{klm}, \{f\}, \mathcal{D}) \mid k,l,m \in \mathbb{N}, f \in C^{FSM}_{klm}\}$, where $\mathcal{D}$ is the trivial distinguishing relation. 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 $\mathcal{T}_{EFSM} = \{(C^{EFSM}_{klmpq}, \{f\}, \mathcal{D}) \mid k,l,m,p,q \in \mathbb{N}, f \in C^{EFSM}_{klmpq}\}$. 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 $\mathcal{T}_{TEFSM} = \{(C^{TEFSM}_{klmpqrs}, \{f\}, \mathcal{D}) \mid k,l,m,p,q,r,s \in \mathbb{N}, f \in C^{TEFSM}_{klmpqrs}\}$. 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 triple $a = (C^{FSM}_{klm}, \{f\}, \mathcal{D}) \in \mathcal{T}_{FSM}$, let us consider the triple $b = (C^{EFSM}_{klm00}, \{f\}, \mathcal{D}) \in \mathcal{T}_{EFSM}$. Let us note that functions in C^{FSM}_{klm} are the same as functions 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 10 to transform triples 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 $e((C^{FSM}_{klm}, \{f\}, \mathcal{D})) = (C^{EFSM}_{klm00}, \{f\}, \mathcal{D})$ and t is the identity function.[10] 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 $e((C^{EFSM}_{klmpq}, \{f\}, \mathcal{D})) = (C^{TEFSM}_{klmpq00}, \{f'\}, \mathcal{D})$ 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, but, in particular, since $e(a)$ follows the form $(C^{TEFSM}_{klmpq00}, \{f'\}, \mathcal{D})$, 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: $t(\mathcal{I}) = \{(i_1\cdots i_n) | ((i_1,d_1)\cdots(i_n,d_n)) \in \mathcal{I}\}$. By considering these definitions of e and t, the requirement imposed by Definition 10 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 $e((C^{EFSM}_{klmpq}, \{f\}, \mathcal{D})) = (C^{FSM}_{(k*p*q)\ lm}, \{f\}, \mathcal{D})$. 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

[10] Let us note that e is also the identify function.

$e(a) = a$). Thus, if t is the identity function then e and t fulfill the requirement of Definition 10 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 labelled 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 labelled by (i, d) explicitly updates the value of all EFSM variables representing the TEFSM clocks according to d. Let $e((C^{TEFSM}_{klmpqrs}, \{f\}, \mathcal{D})) = (C^{EFSM}_{k\ (l*s)\ mp\ (max(q,s))}, \{f\}, \mathcal{D})$, 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 5, for all $a = (C^{FSM}_{klm}, \{f\}, \mathcal{D}) \in \mathcal{T}_{FSM}$ 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 FSM-EFSM-TEFSM example exploits the idea of mapping a computation into another: Due to the constraints imposed to EFSMs and EFSMs, it turns out that each computation formalism can simulate the other two ones (even though stimulating one of these systems consists in producing sequences of *pairs* of the form (input,time)). Moreover, for each triple $(C, E, \mathcal{D})$ belonging to each testing problem of Example 8, we have that C is a *finite* set of functions. At this point it is worth to point out that, in general, the testing reduction does *not* consist in mapping a computation formalism into another, but in mapping the *border* between correctness and incorrectness from a problem to another. Next we show an example where making the testing reduction requires mapping the correctness border, not computation formalisms. Moreover, all considered computation formalisms are *infinite*.

Example 9. Let $\mathcal{D}$ be the trivial distinguishing, C_{TM} represent all deterministic terminating Turing Machines (TMs) 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 triples where we have to distinguish some TM M from all TMs answering as M for at most $k \in \mathbb{N}$ inputs. Formally, let $\mathcal{T}_{TM}$ be defined as follows: $\mathcal{T}_{TM} = \{(\{f\} \cup C^{f,k}_{TM}, \{f\}, \mathcal{D}) | f \in C_{TM}, k \in \mathbb{N}\}$ where we have $C^{f,k}_{TM} = \{f' | f' \in C_{TM}, f'$ gives the same answer as f for at most k words$\}$. Similarly, let $\mathcal{T}_{FA} = \{(\{f\} \cup C^{f,k}_{FA}, \{f\}, \mathcal{D}) | f \in C_{FA}, k \in \mathbb{N}\}$ where $C^{f,k}_{FA} = \{f' | f' \in C_{FA}, f'$ gives the same answer as f for at most k words$\}$. Note that any triple in $\mathcal{T}_{TM}$ or $\mathcal{T}_{FA}$ is uniquely characterized by f and k. We have $\mathcal{T}_{TM} \leq_F \mathcal{T}_{FA}$. In particular, functions e and t considered in Definition 10 can be defined as follows. First, $e : \mathcal{T}_{TM} \longrightarrow \mathcal{T}_{FA}$ is such that $e((\{f\} \cup C^{f,k}_{TM}, \{f\}, \mathcal{D})) = (\{f'\} \cup C^{f',k}_{FA}, \{f'\}, \mathcal{D})$,

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 $p = (\{f\} \cup C_{TM}^{f,k}, \{f\}, \mathcal{D}) \in \mathcal{T}_{TM}$. We have that $\mathcal{I}$ is a complete test suite for $e(p) = (\{f'\} \cup C_{FA}^{f',k}, \{f'\}, \mathcal{D})$ 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 $p = (\{f\} \cup C_{TM}^{f,k}, \{f\}, \mathcal{D}) \in \mathcal{T}_{TM}$. So, $\mathcal{T}_{TM} \leq_F \mathcal{T}_{FA}$. For all $q = (\{f\} \cup C_{FA}^{f,k}, \{f\}, \mathcal{D}) \in \mathcal{T}_{FA}$, 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 $\mathcal{T}_{FA} \leq_F \mathcal{T}_{TM}$. □

4 Conclusions and future work

Formal Testing Techniques have reached a high level of maturity during the last years. However, some common roots allowing 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 the 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 to *reduce* a testing problem into another. From a theoretical point of view, these techniques allow to relate testing problems with each other as well as to classify them. From a practical point of view, they allow us to determine the (im-)possibility to find complete test suites in different scenarios, as well as 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 (many examples have been presented). Going one step further, the proposed general properties could be particularized and refined for specific computation formalisms.

More generally, properties presented in Section 3 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 to organize 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, *probabilistic* testability, *heuristic* testability, 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) Java function f from naturals to naturals 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`. However, if tests may 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 and their corresponding adaptive counterparts should be studied. Finally, we wish to study the applicability of the proposed testability notions to the related problem of *learnability*.

Acknowledgements I 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 CONCUR'09 for their interesting comments.

References

1. 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.
2. 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.
3. J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North Holland, 1976.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
10. 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.
11. R.M. Hierons. Verdict functions in testing with a fault domain or test hypotheses. *ACM Transactions on Software Engineering and Methodology*, 18(4), 2009.
12. 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.
13. 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.

14. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
15. 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.
16. 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.
17. 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.
18. I. Rodríguez. A general testability theory. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, LNCS 5710*, pages 572–586. Springer, 2009.
19. 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.
20. 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.
21. 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.
22. J. Tretmans. Testing concurrent systems: A formal approach. In *10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.

Appendix: Proofs

Proof of the claim "*The set of all sequences of rational-timed inputs is countable*" used in Example 3:

Following our usual notation convention, let I' be the finite set of basic inputs. We have to count the set of all sequence of *rational-timed* inputs, that is, the set $V = \{(i_1, \frac{q_1}{q_1'}), \ldots, (i_n, \frac{q_n}{q_n'}) \mid n \in \mathbb{N}, i_1, \ldots, i_n \in I', q_1, q_1', \ldots, q_n, q_n' \in \mathbb{N}\}$. For all $\sigma = i_1 \cdot \ldots \cdot i_n \in I'^*$, let $I_\sigma = \{(i_1, \frac{q_1}{q_1'}) \cdot \ldots \cdot (i_n, \frac{q_n}{q_n'}) \mid q_1, q_1', \ldots, q_n, q_n' \in \mathbb{N}\}$. For all $\sigma \in I'^*$, we can count the set I_σ by using a surjective function $f_\sigma : \mathbb{N} \to I_\sigma$ defined as follows. Let p_j denote the j-th prime number. Then, $f_\sigma(j) = (i_1, \frac{e_1}{e_2}) \cdot \ldots \cdot (i_n, \frac{e_{2n-1}}{e_{2n}})$ if $j = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_{2n-1}^{e_{2n-1}} \cdot p_{2n}^{e_{2n}}$; otherwise, $f_\sigma(j) = (i_1, 1) \cdot \ldots \cdot (i_n, 1)$. Let $g_\sigma : \mathbb{N} \to I_\sigma$ be defined in such a way that $g_\sigma(0) = (i_1, 1) \cdot \ldots \cdot (i_n, 1)$ and $g_\sigma(j)$ with $j \geq 1$ returns the j-th element in the sequence $f_\sigma(0), f_\sigma(1), f_\sigma(2), \ldots$ being different from $(i_1, 1) \cdot \ldots \cdot (i_n, 1)$. Let l_j denote the j-th input sequence in I'^* in lexicographic order. Let us note that $V = \{I_\sigma | \sigma \in I'^*\}$. We can cover all elements in V in the following order. First we take $g_{l_0}(0)$. Next we add $g_{l_0}(1)$ and $g_{l_1}(0)$. Next we consider $g_{l_0}(2)$ and $g_{l_1}(1)$,

and $g_{l_2}(0)$. Next we add $g_{l_0}(3)$ and $g_{l_1}(2)$, $g_{l_2}(1)$, $g_{l_3}(0)$, and so on. In this way, we count the set V of all sequences of rational-timed inputs.

Proof of the property $(C_2, E_2, \mathcal{D}) \in$ `Class II` given in Example 4:

We prove that there exists a bijective function $h : \mathbb{N} \longrightarrow C_2$ such that the triple $(C_2, E_2, \mathcal{D})$ is unboundedly-approachable by finite testing through h. First we 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' (ties are solved by applying any arbitrary criterium). Let $h : \mathbb{N} \longrightarrow C_2$ be a bijective function such that the sequence $\alpha = [h(1), h(2), \ldots]$ is iteratively constructed from an empty sequence as follows: (step 1) for each possible combination of responses to input sequences $\sigma_a = a$ and $\sigma_b = b$ (that is, for responses $f(a) = 0$ and $f(b) = 0$; responses $f(a) = 0$ and $f(b) = 1$; and so on), we add to α the smallest function f providing exactly these responses; (step 2) we do the same for input sequences $\{aa, ab, ba, bb\}$, with the following exception: If, for some combination of responses, the function to be included in α was already included before, then no function is added now for this combination of responses; (step 3) we consider 3-length input sequences with the same exception; and so on. It is easy to see that (a) for all $f \in C_2$, f appears in α exactly once, and (b) after completing each step l, that is, after adding the functions corresponding to a new set of input sequences, the total number of functions in α providing each possible combination of responses to $\sigma_1, \ldots, \sigma_n$, where $\{\sigma_1, \ldots, \sigma_n\}$ is the set of inputs considered in any step l' with $l' \leq l$, is the same for all combinations.

We prove $(C_2, E_2, \mathcal{D})$ is unboundedly-approachable by finite testing through h. Recall that $E_2 = \{f_1\}$ consists of a single function f_1. Let $0 \leq \varepsilon < 1$. We will find a test suite providing a distinguishing rate higher than or equal to ε for some $C_2 \downarrow_q h$ with $q \in \mathbb{N}$. Besides, we will see that for all $q' > q$ there exists $q'' \geq q'$ such that the *same* suite provides that distinguishing rate for $C_2 \downarrow_{q''} h$. For all $l \in \mathbb{N}$, let $\mathcal{I}^l = \{i_1 \cdots i_l | \forall\ 1 \leq m \leq l : i_m \in \{a, b\}\}$, and let $\mathcal{P}^l = \{P_1^l, \ldots, P_{t_l}^l\}$ be a partition of C_2 where, for all $1 \leq j \leq t_l$, all functions in P_j^l provide the same responses for all tests in $\mathcal{I}^l$, i.e., for all $f, f' \in P_j^l$ and for all $\sigma \in \mathcal{I}^l$ we have $f(\sigma) = f'(\sigma)$. Let $P_1^l \in \mathcal{P}$ be the (unique) set of $\mathcal{P}$ such that for all $f \in P_1^l$ and for all $\sigma \in \mathcal{I}$ we have $f(\sigma) = f_1(\sigma)$. Since the only correct function is $f_1 \in E_2$, testing E_2 against C_2 consists in distinguishing all pairs in the set $\{(f_1, f) | f \in C_2 \backslash \{f_1\}\}$. Let $q \in \mathbb{N}$ be the minimum natural such that (a) $f_1 \in C_2 \downarrow_q h$ and (b) for some $l \in \mathbb{N}$ with $\frac{1}{|\mathcal{P}^l|} < 1 - \varepsilon$, q is equal to the number of functions belonging to α right after applying step l as described in the previous paragraph (i.e. right after adding the functions corresponding to the set of input sequences $\mathcal{I}^l$). If E_2 and C_2 are restricted to the first q functions of C_2, then testing consists in distinguishing all pairs in the set $\{(f_1, f) | f \in (C_2 \downarrow_q h) \backslash \{f_1\}\}$. Thus, $\texttt{d-rate}\left(\mathcal{I}^l, C_2 \downarrow_q h, E_2 \downarrow_q h, \mathcal{D}\right) = 1 - \frac{|(C_2 \downarrow_q h) \cap P_1^l|}{q}$, i.e. one minus the proportion of pairs *not* distinguished by $\mathcal{I}^l$. By the definition of h, this proportion is equal

to $\frac{1}{|\mathcal{P}^l|}$. Thus, $\texttt{d-rate}\left(\mathcal{I}^l, C_2 \downarrow_q h, E_2 \downarrow_q h, \mathcal{D}\right) > \varepsilon$. For all $q' > q$, let q'' with $q'' \geq q'$ be the number of functions belonging to α after completing some step l' with $l' > l$. We have $\texttt{d-rate}\left(\mathcal{I}^l, C_2 \downarrow_{q''} h, E_2 \downarrow_{q''} h, \mathcal{D}\right) = 1 - \frac{|(C_2 \downarrow_{q''} h) \cap P_1^l|}{q''}$. The proportion of pairs not distinguished by $\mathcal{I}^l$ in $C_2 \downarrow_{q''} h$ is the same as in $C_2 \downarrow_q h$, that is, $\frac{1}{|\mathcal{P}^l|}$. Thus, we have $\texttt{d-rate}\left(\mathcal{I}^l, C_2 \downarrow_{q''} h, E_2 \downarrow_{q''} h, \mathcal{D}\right) > \varepsilon$ again.

Proof of Lemma 1:

We consider (1). By reductio ad absurdum, let us suppose that there exists $f' \in C \backslash E$ such that for all $i \in \mathcal{I}$ there exists $o' \in f'(i)$ and $(i, o) \in \mathcal{G}$ with $o \not\mathcal{P}\, o'$. Since $f \in E$ and for all $(i, o) \in \mathcal{G}$ we have $o \in f(i)$, this means that there exist $f \in E$ and $f' \in C \backslash E$ such that for all $i \in \mathcal{I}$ there exists $o \in f(i)$ and $o' \in f'(i)$ with $o \not\mathcal{P}\, o'$. This contradicts the fact that $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$.

Proof of Lemma 2:

Properties (a) and (b) are straightforward. Next we prove (c). 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 for all $i \in I$ we have $o_1 \not\mathcal{P}\, o_2$ for some $o_1 \in f(i)$ and $o_2 \in f(i)$, for each pair $(f, f') \in C$ there exists some input $i \in I$ allowing to distinguish f and f', that is, there exists $i \in I$ such that for all $o \in f(i)$ and $o' \in f'(i)$ we have $o \,\mathcal{D}\, o'$. Thus, a finite set containing, for each pair (f, f'), an input allowing to distinguish f and f', is a finite complete test suite for C, E, and $\mathcal{D}$. Thus, $(C, E, \mathcal{D}) \in \texttt{Class I}$.

We consider (d). Let $\mathcal{H}$ be the set of all pairs (i, f') such that $f' \in C \backslash E$ and for all $i' \neq i$ we have $f(i') = f'(i')$. Let us suppose that there exists $(i, f') \in \mathcal{H}$, $o \in f(i)$, and $o' \in f'(i)$ such that $o \not\mathcal{P}\, o'$. Then, f and f' are not distinguishable, so by Lemma 2 (a) we have $(C, E, \mathcal{D}) \notin \texttt{Class IV}$. Otherwise, for each $(i, f') \in \mathcal{H}$ we have that f and f' are distinguished only by i. Hence, if $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$ then all inputs appearing in $\mathcal{H}$ are in $\mathcal{I}$, that is $\{i | (i, f') \in \mathcal{H}\} \subseteq \mathcal{I}$. For each $i \in I$ there exists $(i, f') \in \mathcal{H}$ and I is infinite, so there does not exist any finite complete test suite for C, E, and $\mathcal{D}$. Thus, $(C, E, \mathcal{D}) \notin \texttt{Class I}$.

Proof of Lemma 3:

We consider (c). First, we prove the implication from left to right. For all $g \in E \backslash \{f\}$ and $g' \in (C \backslash \{f\}) \backslash (E \backslash \{f\})$ we have $g \in E$ and $g' \in C \backslash E$. Since $\mathcal{I}$ is a complete test suite for C, E, and $\mathcal{D}$, there exists $i \in \mathcal{I}$ such that i distinguishes g and g'. Thus, $\mathcal{I}$ is a complete test suite for $C \backslash \{f\}$, $E \backslash \{f\}$, and $\mathcal{D}$.

We consider the implication from right to left. Let $\mathcal{I}$ be a complete test suite for $C \backslash \{f\}$, $E \backslash \{f\}$, and $\mathcal{D}$, and let us suppose that $\mathcal{I}$ is not a complete test suite for C, E, and $\mathcal{D}$. This implies that f is involved in a non-distinguishable pair of functions of E and $C \backslash E$, that is, there exists $f' \in C \backslash E$ such that for all $\mathcal{I}$ there exist $o \in f(i)$ and $o' \in f'(i)$ with $o_1 \not\mathcal{P}\, o_2$. Let B be the set of all inputs $i \in I$ such

that for all $o \in f(i)$ and $o' \in f(i)$ we have $o \mathcal{D} o'$. We are assuming that there exists $(g, g', B') \in Q$ with $g \neq f$ such that $B' \subseteq B$ and $B' \neq \emptyset$. In particular, this implies $B \neq \emptyset$. Since $\mathcal{I}$ is a complete test suite for $C \backslash \{f\}$, $E \backslash \{f\}$, and $\mathcal{D}$, there exists $i \in \mathcal{I}$ such that for all $o \in g(i)$ and $o' \in g'(i)$ we have $o \mathcal{D} o'$. We have $i \in B'$, so $i \in B$. Thus, i distinguishes f and f', which makes a contradiction.

Proof of Proposition 1:

We consider two possibilities: (a) there exist $f \in E$ and $f' \in C \backslash E$ such that for all $i \in I$ we have $o_1 \not\mathcal{D} o_2$ for some $o_1 \in f(i)$ and $o_2 \in f'(i)$; and (b) there do not exist such f and f'. In case (a) we cannot distinguish f and f', so by Lemma 2 (a) we have $(C, E, \mathcal{D}) \notin$ `Class IV` and thus $(C, E, \mathcal{D}) \notin$ `Class I`. In case (b), for all $A \in \mathcal{A}$ there exist $f \in E$ and $f' \in C \backslash E$ that are distinguished with an input $i \in A$ (and are not distinguished by any input $i' \notin A$). Since A is disjoint with all other set $A' \in \mathcal{A}$, a complete test suite must include at least one input from each set $A \in \mathcal{A}$. Thus, if $\mathcal{A}$ is infinite then $(C, E, \mathcal{D}) \notin$ `Class I`, and if $\mathcal{A}$ is finite and there exists a finite complete test suite $\mathcal{I}$ then we must have $|\mathcal{I}| \geq |\mathcal{A}|$.

Proof the claim "*We may have $(C, E, \mathcal{D}) \notin$ `Class I` even if there does not exist any infinite set $\mathcal{A}$ fulfilling conditions (1) and (2) of Proposition 1***" given in Example 7:**

We prove that we may have $(C, E, \mathcal{D}) \notin$ `Class I` even if there does not exist any infinite set $\mathcal{A}$ fulfilling conditions (1) and (2) of Proposition 1 (that is, the longest set fulfilling (1) and (2) is finite). Let C be a computation formalism, $E \subseteq C$ be a specification such that E and $C \backslash E$ are *infinite* sets, and $\mathcal{D}$ be the trivial distinguishing relation. Inputs will be decorated with the name of two correct and two incorrect *functions*. Specifically, $I = \left\{ i_{x,y}^{x',y'} \mid x, y \in E \ \wedge \ x', y' \in C \backslash E \right\}$. 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 $q \in C$, 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, the input $i_{f,g}^{f',g'}$ distinguishes h and h' if $h \in \{f, g\}$ and $h' \in \{f', g'\}$ because we have $h\left(i_{f,g}^{f',g'}\right) \cap h'\left(i_{f,g}^{f',g'}\right) = \emptyset$. Otherwise, i.e. if $h \notin \{f, g\}$ or $h' \notin \{f', g'\}$, we have $h\left(i_{f,g}^{f',g'}\right) \cap h'\left(i_{f,g}^{f',g'}\right) \neq \emptyset$, so $i_{f,g}^{f',g'}$ does not distinguish h and h'. Any set of inputs A fulfilling condition (2) of Proposition 1 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 (2). Since condition (1) of Proposition 1 requires all sets of $\mathcal{A}$ to be pairwise disjoint, conditions (1) and (2) 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 E, C, and $\mathcal{D}$ 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, $(C, E, \mathcal{D}) \notin$ `Class I`.

Proof of Proposition 2:

We consider the implication from left to right. If $(C, E, \mathcal{D}) \in$ `Class I` then there exists a finite complete test suite $\{i_1, \ldots, i_n\}$ for C, E, and $\mathcal{D}$. 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 $\bigcap_{B \in A_j} B \neq \emptyset$. Since $\{i_1, \ldots, i_n\}$ is a finite complete test suite, by Lemma 3 (a) we have that for all $B \in G$ we have $B \cap \{i_1, \ldots, i_n\} \neq \emptyset$. So, for some $1 \leq j \leq n$ we have $i_j \in B \cap \{i_1, \ldots, i_n\}$. Thus, for all $B \in G$ we have $B \in A_j$ for some $1 \leq j \leq n$, and we have $\bigcup_{1 \leq j \leq n} A_j = G$.

The implication from right to left 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 $\bigcap_{B \in A_j} B \neq \emptyset$.

Proof of Lemma 4:

We consider the implication from left to right. By contrapositive, let us suppose that there exist $f \in E$ and $f' \in C\backslash E$ such that $f \notin H$, $f' \notin H$, and for all $i \in \mathcal{I}$ there exist $o \in f(i)$, $o' \in f'(i)$ with $o \not\mathcal{D} o'$. Since $f \in E\backslash H$ and $f' \in (C\backslash H)\backslash(E\backslash H)$, $\mathcal{I}$ is not a complete test suite for $C\backslash H$, $E\backslash H$, and $\mathcal{D}$. Thus, H does not enable $\mathcal{I}$ for $(C, E, \mathcal{D})$.

We check the implication from right to left. Again by contrapositive, let us suppose that H does not enable $\mathcal{I}$ for $(C, E, \mathcal{D})$. Since $\mathcal{I}$ is not a complete test suite for C, E, and $\mathcal{D}$, by the definition of the *enable* relation we have that $\mathcal{I}$ is not a complete test suite for $C\backslash H$, $E\backslash H$, and $\mathcal{D}$. This means that there exist $f \in E\backslash H$, $f' \in (C\backslash H)\backslash(E\backslash H)$ such that for all $i \in \mathcal{I}$ there exists $o \in f(i)$ and $o' \in f'(i)$ with $o \not\mathcal{D} o'$. Since $C\backslash H \subseteq C$ and $E\backslash H \subseteq E$, we have $f \in E$ and $f' \in C\backslash E$. Moreover, since $f \in E\backslash H$ and $f' \in (C\backslash H)\backslash(E\backslash H)$, we also have $f \notin H$ and $f' \notin H$. Thus, there exist $f \in E$ and $f' \in C\backslash E$ with $f \notin H$ and $f' \notin H$ such that for all $i \in \mathcal{I}$ there exists $o \in f(i)$ and $o' \in f'(i)$ with $o \not\mathcal{D} o'$.

Proof of Theorem 1:

Let $|I| = s_I$, $|O| = s_O$, and $|C'| = s_{C'}$. Let us note that the representation size of C' and E' is in $\mathcal{O}(s_{C'} \cdot s_I \cdot s_O)$ and the representation size of $\mathcal{D}$ is in $\mathcal{O}(s_O^2)$. First, let us prove `MCS` $\in$ `NP`. Given C', E', I, O, $\mathcal{D}$, K as input of `MCS`, 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, E, and $\mathcal{D}$ requires polynomial time. Since $\mathcal{I} \subseteq I$, we calculate $|\mathcal{I}|$ in time $\mathcal{O}(s_I)$. 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 $o \; \mathcal{D} \; o'$. This can be done in time $\mathcal{O}(s^2_{C'} \cdot s_I \cdot s^2_O)$. Thus, MCS $\in$ NP.

In order to prove MCS $\in$ NP-complete, we polynomially reduce an NP-complete problem to MCS. We will consider the *Minimum 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 $|J'| \leq K$.[11]

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, and $\mathcal{D}$ from them in polynomial time, where C' and E' will represent some computation formalism C and some specification E, respectively. The idea of the transformation from Minimum Set Cover to MCS is the following: We construct an instance of MCS 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 MCS 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 $\alpha^j_l \in f_j(i_l)$ for some unique output α^j_l. We will use the trivial distinguishing relation, i.e. $o_1 \; \mathcal{D} \; o_2$ iff $o_1 \neq o_2$. Thus, if only the unique outputs α^j_l 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 $\beta_j \in f_j(i_l)$ and $\beta_j \in f'(i_l)$ for some output β_j. Since $f_j(i_l) \cap f'(i_l) \neq \emptyset$, we have that 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 $C', E', I, O, \mathcal{D}$ from J, S as follows. Each tuple $(outs^j_1, \ldots, outs^j_k) \in E'$ will extensionally denote the behavior of a function $f_j \in E$ for all inputs. In

[11] We could consider another problem, such as the *Minimum Test Collection* problem. Despite of its name, this problem is very different to MCS and thus is not a good choice to construct a polynomial reduction from an NP-complete problem to MCS. 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.

particular, for all $1 \leq l \leq k$ we will consider $f_j(i_l) = outs_l^j$. Similarly, a tuple $(outs'_1, \ldots, outs'_k) \in C'$ will denote the behavior of the (single) incorrect function $f' \in C \backslash E$. Formally, we consider:

- $C' = \{(outs_1^j, \ldots, outs_k^j) | 1 \leq j \leq n\} \bigcup$
  $\{(outs'_1, \ldots, outs'_k) | 1 \leq j \leq n\}$
  where:
  - for all $1 \leq j \leq n$ and $1 \leq l \leq k$ we have $outs_l^j = \{\alpha_l^j\} \cup \{\beta_j | x_j \notin S_l\}$,
  - for all $1 \leq l \leq k$ we have $outs'_l = \{\alpha'_l\} \cup \{\beta_j | x_j \notin S_l \ \wedge \ 1 \leq j \leq n\}$.
- $E' = \{(outs_1^j, \ldots, outs_k^j) | (outs_1^j, \ldots, outs_k^j) \in C'\}$
- $I = \{i_1, \ldots, i_k\}$.
- $O = \{\alpha_l^j | 1 \leq j \leq n \ \wedge \ 1 \leq l \leq k\} \cup$
  $\{\alpha'_l | 1 \leq l \leq k\} \cup \{\beta_j | 1 \leq j \leq n\}$
- $\mathcal{D} = \{(o_1, o_2) | o_1, o_2 \in O \ \wedge \ o_1 \neq o_2\}$.

Assuming $|J| = s_J$ and $|S| = s_S$, it is easy to check that the representation size of C' and E' is in $\mathcal{O}(s_S^2 \cdot s_J)$ (s_S pairs, s_J inputs for each pair, and $s_S + 1$ outputs for each input in the worst case). Representing I requires a size in $\mathcal{O}(s_S)$, O can be represented in $\mathcal{O}(s_S \cdot s_J)$, and $\mathcal{D}$ requires $\mathcal{O}(s_S^2 \cdot s_J^2)$. Thus, the constructed instance of MCS 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, and $\mathcal{D}$ 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, and $\mathcal{D}$. 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 $f_j(i_l) \cap f'(i_l) \neq \emptyset$. 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_j. So, for all $i_l \in \{i_{a_1}, \ldots, i_{a_K}\}$ we have $f_j(i_l) \cap f'(i_l) = \{\beta_j\}$. By the construction of C' and E' from J and S, if $\beta_j \in f_j(i_l)$ then $x_j \notin S_l$. Thus, for all $S_l \in \{S_{a_1}, \ldots, S_{a_K}\} = J'$ we have $x_j \notin S_l$. Thus, $\bigcup_{S' \in J'} S' \neq S$ and we have a contradiction.

$\Longleftarrow$: We assume that $\mathcal{I} = \{i_{a_1}, \ldots, i_{a_K}\}$ is a complete test suite for C, E, and $\mathcal{D}$, 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 $\beta_j \in f_j(i_l)$ and $\beta_j \in f'(i_l)$. 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, and $\mathcal{D}$. Hence, we have a contradiction.

Proof of Theorem 2 (a):

We prove MFR $\in$ P.[12] From an instance of MFR, let us construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows: $\mathcal{V} = C'$ and for all $f_1 \in E'$ and $f_2 \in C' \backslash E'$ such that

[12] For the sake of notation simplicity, sometimes we will not distinguish C' and E' from the sets they represent, i.e. C and E, respectively.

$\texttt{di}\,(f_1, f_2, \mathcal{I})$ does *not* hold we have $(f_1, f_2) \in \mathcal{E}$. By Lemma 4, a set $R \subseteq C'$ is such that $\mathcal{I}$ is a complete test suite for $(C'\backslash R, E'\backslash R, \mathcal{D})$ iff, 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 $f_1 \in R$ or $f_2 \in R$. 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 $(C'\backslash R, E'\backslash R, \mathcal{D})$ is equivalent to removing K or less vertexes from $\mathcal{G}$ in such a way that all edges are lost. That is, MFR is equivalent to finding a subset of vertexes $\mathcal{V}' \subseteq \mathcal{V}$ such that $|\mathcal{V}'| \leq K$ and the subgraph $\mathcal{G}'$ of $\mathcal{G}$ induced by $\mathcal{V}\backslash\mathcal{V}'$ has no edges, that is, $\{(v, v')|(v, v') \in \mathcal{E}, \{v, v'\} \subseteq \mathcal{V}\backslash\mathcal{V}'\} = \emptyset$. Let us consider the *Minimum 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 MFR requires that, for each edge, at least one of the endpoints of the edge is removed, we can solve MFR in terms of the Minimum 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 MFR 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 *Minimum Vertex Cover* is NP-complete in general graphs, König's theorem [3] states that it is equivalent to the *Maximum Matching* problem if the graph is *bipartite* (and $\mathcal{G}$ is so). The Maximum Matching 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 [12]. Hence, MFR 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 $\texttt{di}\,(f_1, f_2, \mathcal{I})$ 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 $o \not\mathcal{P} o'$. Thus, the cost of constructing $\mathcal{G}$ is in $\mathcal{O}(|C'|^2 \cdot |\mathcal{I}| \cdot |\mathcal{O}|^2)$. The cost of executing the Hopcroft-Karp algorithm is, in terms of the MFR problem instance, in $\mathcal{O}(\sqrt{|C'|} \cdot |C'|^2) = \mathcal{O}(|C'|^{5/2})$. Thus, MFR can be solved in time $\mathcal{O}(|C'|^{5/2} + |C'|^2 \cdot |\mathcal{I}| \cdot |\mathcal{O}|^2)$ and we have $\texttt{MFR} \in \texttt{P}$.

Proof of Theorem 2 (b):

We prove $\texttt{MFH} \in \texttt{NP-complete}$.[13] Let $s_I = |I|$, $s_O = |O|$, $s_{C'} = |C'|$, and $s_{\mathcal{H}} = |\mathcal{H}|$. First we prove $\texttt{MCS} \in \texttt{NP}$. Given C', E', I, O, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{H}$, K as input of MFH, 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 $(C'\backslash(\bigcup_{H \in R} H), E'\backslash(\bigcup_{H \in R} H), \mathcal{D})$. 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 $\mathcal{O}(s_{\mathcal{H}} \cdot s_{C'}^2)$. 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

[13] For the sake of notation simplicity, sometimes we will not distinguish C' and E' from the sets they represent, i.e. C and E, respectively.

these sets. Since for all $H \in R$ we have $H \subseteq C'$, these subtractions can be done in time $\mathcal{O}(s_{\mathcal{H}} \cdot s_{C'}^2)$. 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 $o \, \mathcal{D} \, o'$. 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 $\mathcal{O}(s_{C'}^2 \cdot s_I \cdot s_O^2)$. Thus, checking whether some given $R \in \mathcal{H}$ fulfills the required conditions can be done in polynomial time. Hence, $\texttt{MCS} \in \texttt{NP}$.

In order to prove $\texttt{MFH} \in \texttt{NP-complete}$, we polynomially reduce an NP-complete problem, `3-SAT`, to `MFH`. 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 $\texttt{props}(\varphi) = \{p_1, \ldots, p_k\}$ 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 $\nu : \texttt{props}(\varphi) \longrightarrow \{\top, \bot\}$. 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 $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K)$ of the `MFH` 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 $(C' \backslash (\bigcup_{H \in R} H), E' \backslash (\bigcup_{H \in R} H), \mathcal{D})$ if and only if φ is satisfiable. Before formally presenting this construction, we give an intuitive explanation. We will construct an `MFH` instance where, for each literal l_k^j of φ, there is a function in E' and a function in $C' \backslash E'$ representing l_k^j. 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 an output α that is given *only* by that function. Thus, if this is the only output produced by a function then this function will be distinguishable from any other function (as we will consider the trivial distinguishing relation: $o_1 \, \mathcal{D} \, o_2$ 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, $\texttt{di}\,(f_1, f_2, \mathcal{I})$ will not hold. Thus, we can make any pair of functions (un-)distinguishable as required for our construction.

In `MFH`, 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 $\mathcal{H}$ an 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 `MFH`, we will consider that *not* 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 `MFH` 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 `MFH` then φ must be satisfiable.

Next we formally construct an `MFH` instance $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K)$ 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 $outs_k^j = \{\alpha_k^j, \beta_y^q, \beta_k^j, \beta_{k_1}'^j, \beta_{k_1}'^j\}$ 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 $outs_k'^j = \{\alpha_k'^j, \beta_y^q, \beta_k'^j, \beta_{k_1}^j, \beta_{k_1}^j\}$ 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 $\{k_1, k_2\} = \{1, 2, 3\}\backslash\{k\}$.
- $E' = \{(outs_k^j) | (outs_k^j) \in C'\}$
- $I = \{i\}$

- $O = \{\alpha_k^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup$
  $\{\alpha_k'^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup$
  $\{\beta_k^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup$
  $\{\beta_k'^j | 1 \leq j \leq n, 1 \leq k \leq 3\} \cup$
  $\{\beta_y^q | 1 \leq q \leq m, y \in \{\top, \bot\}\}$
- $\mathcal{D} = \{(o_1, o_2) | o_1, o_2 \in O \ \wedge \ o_1 \neq o_2\}$
- $\mathcal{I} = \{i\}$
- $\mathcal{H} = \{\{(outs_k^j), (outs_k'^j)\} | 1 \leq j \leq n, 1 \leq k \leq 3\}$
- $K = 4 \cdot n$

We show that the construction of $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K)$ 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 $\mathcal{D}$ is extensionally defined, we have $|\mathcal{D}| = |O|^2$, and we have $|\mathcal{H}| = |C'|/2$. In conclusion, $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K)$ can be constructed from φ in polynomial time.

Let us show that the answer of `MFH` to $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K)$ 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 $(C' \backslash (\bigcup_{H \in R} H), E' \backslash (\bigcup_{H \in R} H), \mathcal{D})$. Then, by Lemma 4, for all $f \in C'$ and $f' \in C' \backslash E'$ such that $\mathtt{di}\,(f, f', \mathcal{I})$ does not hold we have either $f \in \bigcup_{H \in R} H$ or $f' \in \bigcup_{H \in R} H$ (or both). Since functions representing *opposite* literals are made undistinguishable in the construction of the `MFH` 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 viceversa. 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 viceversa. 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 viceversa. 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 triple $(C'\backslash(\bigcup_{H\in R} H), E'\backslash(\bigcup_{H\in R} H), \mathcal{D})$. 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 $R = \{H | H \in \mathcal{H}, H \cap Q = \emptyset\}$. 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 4, 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 $(C'\backslash(\bigcup_{H\in R} H), E'\backslash(\bigcup_{H\in R} H), \mathcal{D})$. By the construction of E' and $C'\backslash E'$, a pair of functions is undistinguishable if either (a) they represent opposite literals or (b) if 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 $(C'\backslash(\bigcup_{H\in R} H), E'\backslash(\bigcup_{H\in R} H), \mathcal{D})$.

Proof of Theorem 2 (c):

We prove $\texttt{MHA} \in \texttt{NP-complete}$.[14] Given C', E', I, O, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{H}$, K as input of MHA, checking whether some given $R \subseteq \mathcal{H}$ is such that $|R| \leq K$ and $\mathcal{I}$ is a complete test suite for $(C'\backslash(\bigcup_{H\in R} H), E'\backslash(\bigcup_{H\in R} H), \mathcal{D})$ requires polynomial time (we can prove it similarly as in the beginning of the proof of Theorem 2 (b)). Thus, $\texttt{MHA} \in \texttt{NP}$. In order to prove $\texttt{MHA} \in \texttt{NP-complete}$, we polynomially reduce an NP-complete problem, the *Minimum Set Cover* problem, to MHA. The Minimum Set Cover, 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 for the Minimum 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 $1 \leq j \leq k$,

[14] For the sake of notation simplicity, sometimes we will not distinguish C' and E' from the sets they represent, i.e. C and E, respectively.

and $K \in \mathbb{N}$, we will construct in polynomial time an MHA instance from them, that is, we construct some C', E', I, O, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{H}$, 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 MHA for $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K')$ is *yes* iff the solution of the Minimum Set Cover for (S, J, K) is *yes* as well. The idea of the transformation from a Minimum Set Cover instance to a MHA 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 a unique output α in response to i, and functions representing the same element $x \in S$ in E' and $C' \backslash E'$ answer a shared output β too. This will make them undistinguishable, because we will consider $o_1 \; \mathcal{D} \; o_2$ iff $o_1 \neq o_2$. For each set $\{x_1, \ldots, x_l\} \in J$, we have an 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 MHA 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 MHA 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 MHA is yes iff the answer of the Minimum Set Cover is yes.

Next we formally construct the MHA instance $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K')$ 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 $outs'^j = \{\alpha'^j, \beta^j\}$
- $E' = \{(outs^j) | (outs^j) \in C'\}$
- $I = \{i\}$
- $O = \{\alpha^j | 1 \leq j \leq n\} \cup \{\alpha'^j | 1 \leq j \leq n\} \cup$
  $\{\beta^j | 1 \leq j \leq n\}$
- $\mathcal{D} = \{(o_1, o_2) | o_1, o_2 \in O \; \wedge \; o_1 \neq o_2\}$
- $\mathcal{I} = \{i\}$

- $\mathcal{H} = \{\{(outs^{j_1}), \ldots, (outs^{j_l})\} | \{x_{j_1}, \ldots, x_{j_l}\} \in J\}$
- $K' = K$

We consider the cost of constructing $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K')$ 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 $\mathcal{D}$ is extensionally defined, we have $|\mathcal{D}| = |O|^2$, and we have $|O| = 3 \cdot |S|$. Besides, $|\mathcal{H}| = |J|$. Thus, $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K')$ can be constructed from (S, J, K) in polynomial time.

Let us show that the answer of MHA to $(C', E', I, O, \mathcal{D}, \mathcal{I}, \mathcal{H}, K')$ 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 $(C' \backslash (\bigcup_{H \in R} H), E' \backslash (\bigcup_{H \in R} H), \mathcal{D})$. By Lemma 4, for all $f_1 \in E'$ and $f_2 \in C' \backslash E'$ such that $\mathtt{di}\,(f_1, f_2, \mathcal{D})$ does not hold, we have that either $f_1 \in \bigcup_{H \in R} H$ or $f_2 \in \bigcup_{H \in R} H$. By the construction of $\mathcal{H}$, for all $H \in \mathcal{H}$ we have $H \subseteq E'$. 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, we have $K' = K$. 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, since $\bigcup_{H \in R} H = E'$, we have $E' \backslash (\bigcup_{H \in R} H) = \emptyset$. Thus, we trivially obtain that $\mathcal{I}$ is a complete test suite for $(C' \backslash (\bigcup_{H \in R} H), E' \backslash (\bigcup_{H \in R} H), \mathcal{D})$.

Proof of Theorem 3:

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 $(C, E, \mathcal{D}) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_1$, if $\mathcal{I}$ is a finite complete test suite for $e(C, E, \mathcal{D}) = (C, E, \mathcal{D})$ then $t(\mathcal{I}) = \mathcal{I}$ is a finite test suite for $(C, E, \mathcal{D})$. 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 for all $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_2$ we have that if $\mathcal{I}$ is a finite complete test suite for $e(C_1, E_1, \mathcal{D}_1)$ then $t(\mathcal{I})$ is a finite complete test suite for $(C_1, E_1, \mathcal{D}_1)$. Besides, there also exist functions $e' : \mathcal{T}_2 \longrightarrow \mathcal{T}_3$ and $t' : 2^{I_3} \longrightarrow 2^{I_2}$ such that for all $(C_2, E_2, \mathcal{D}_2) \in \mathcal{T}_2$ and $\mathcal{I} \subseteq I_3$ we have that if $\mathcal{I}$ is a finite complete test suite for $e'(C_2, E_2, \mathcal{D}_2)$ then $t'(\mathcal{I})$ is a finite complete test suite for $(C_2, E_2, \mathcal{D}_2)$. Let $e'' = e' \circ e$ and $t'' = t \circ t'$.

Let $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$ and $\mathcal{I}$ be a finite complete test suite for $e''(C_1, E_1, \mathcal{D}_1)$. This implies that $t'(\mathcal{I})$ is a complete test suite for $e(C_1, E_1, \mathcal{D}_1)$. This implies that $t(t'(\mathcal{I}))$ is a complete test suite for $(C_1, E_1, \mathcal{D}_1)$. Thus, we can use functions e'' and t'' to make required transformations between $\mathcal{T}_1$ and $\mathcal{T}_2$, and we have $\mathcal{T}_1 \leq_F \mathcal{T}_3$.

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 for all $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$ and $\mathcal{I} \subseteq I_2$ we have that, if $\mathcal{I}$ is a finite complete test suite for $e(C_1, E_1, \mathcal{D}_1)$, then $t(\mathcal{I})$ is a finite complete test suite for $(C_1, E_1, \mathcal{D}_1)$. Besides, let $f : \mathcal{T}_2 \longrightarrow 2^I$ be a finite suite derivation for $\mathcal{T}_2$. We know that f is computable, and for all $(C_2, E_2, \mathcal{D}_2) \in \mathcal{T}_2$ we have that $f(C_2, E_2, \mathcal{D}_2)$ is a finite complete test suite for $(C_2, E_2, \mathcal{D}_2)$. Thus, given $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$, we have that $t(f(e(C_1, E_1, \mathcal{D}_1)))$ is a finite complete test suite for $(C_1, E_1, \mathcal{D}_1)$. The composition of computable functions is computable, so $h = t \circ f \circ e$ 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 $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$ as follows. Since $\mathcal{T}_2 \subseteq$ `Class I`, there exists a finite complete test suite for $e(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_2$. By applying function t, this suite can be transformed into a finite complete test suite for $(C_1, E_1, \mathcal{D}_1) \in \mathcal{T}_1$. Thus, $\mathcal{T}_1 \subseteq$ `Class I`.