Tests and Refutation

Mohammad Torabi Dashti and David Basin

Department of Computer Science, ETH Zurich

Abstract. The purpose of testing a system with respect to a requirement is to refute the hypothesis that the system satisfies the requirement. We build a theory of tests and refutation based on the elementary notions of satisfaction and refinement. We use this theory to characterize the requirements that can be refuted through black-box testing and, dually, verified through such tests. We consider refutation in finite time and obtain the well-known finite falsifiability of hyper-safety temporal requirements as a special case. We extend our theory with computational constraints and separate refutation from enforcement in the context of temporal hyper-properties. Overall, our theory provides a basis to analyze the scope and reach of black-box tests and to bridge results from areas including testing, verification, and enforcement.

1 Introduction

Testing is a widely adopted quality-assurance activity and there is a general agreement as to its purpose and importance. However, a solid understanding of testing's strength and limitations is lacking, despite the manifest importance of this topic. For instance, it is commonly agreed upon that the purpose of testing a system with respect to a requirement is to refute the hypothesis that the system satisfies the requirement [6, 15]. Yet, existing testing theory is inadequate for answering basic questions such as: which class of requirements are refutable, given a class of tests? Or, which class of tests, if any, can refute a class of requirements? The need for research advances here is imperative as current analytic frameworks for testing are incapable of even articulating, let alone answering, these fundamental questions in a satisfactory manner. We show how this can be done for black-box testing, the most basic system analysis technique, by presenting a theory of tests and refutation that fully characterizes the class of refutable requirements and provides a foundation for bridging results in testing with other related disciplines.

We start with an abstract model of systems and requirements (§2) and introduce two types of requirements: obligations and prohibitions (§3). A requirement is an obligation if it obliges the systems to exhibit certain (desired) behaviors, and it is a prohibition if it prohibits the systems from exhibiting (undesired) behaviors. We show that these two requirement types admit a straightforward order-theoretic characterization. Namely, given a refinement (or abstraction) partial-order on a set of systems, the satisfaction of an obligation is abstraction-closed, and for a prohibition it is refinement-closed. We then turn to black-box tests (§4).

A system is a black-box if we can observe its input and output, but cannot observe *how* the latter is produced from the former. Therefore, a tester can analyze such a system only by interacting with it through its interface. In black-box testing, sometimes

called "testing by sampling" [6], testing a system amounts to inspecting a sample of its behaviors. The sample obtained through tests can be seen as a refinement of the system under test, a notion we make precise in the following sections. All a tester learns by sampling is that a system exhibits certain behaviors. From this, the tester cannot infer that the system does not exhibit other behaviors as well. Such a conclusion could only be justified through the sample's exhaustiveness, which black-box testing alone cannot establish. A requirement is therefore refutable through tests if, for any system that violates the requirement, the hypothesis that the system satisfies the requirement can be refuted by inspecting a refinement of the system. It follows that a requirement whose violation is contingent upon demonstrating the absence of behaviors cannot be refuted through black-box testing. Based on this, we prove that any refutable requirement is a prohibition, and all non-trivial obligations are irrefutable (§5). We then define the notion of verification dual to refutation, and show that any verifiable requirement is an obligation and that non-trivial prohibitions cannot be verified through tests (§6).

Contributions. We present a theory for reasoning about the strength and the limitations of black-box testing. Our theory has minimal formal machinery, which gives rise to direct, elementary proofs (Appendix A). We use the theory to prove new results and to obtain known results as special cases.

We fully characterize the requirements that can be refuted and those that can be verified through black-box tests. This characterization augments, and in some cases rectifies, the folkloric understanding that exists in the community. For example, we highlight the fundamental role that determinacy assumptions play in making sense of day-to-day black-box tests.

Our theory is abstract. Extending it to account for refutation in finite time and refutation under computational constraints is therefore immediate. We present different applications of our theory of finite refutability (§7). We demonstrate that the well-known finite falsifiability of hyper-safety temporal requirements [4] can be derived as a special case in our theory. Moreover, we explicate the relationship between finite refutability and system self-composition [3, 4], a central technique in information flow analysis. Finally, we use our characterization to separate refutability from enforceability: we show that any enforceable temporal requirement is refutable, but refutable requirements need not be enforceable; we give a precise definition of the notion of enforceable requirements in the following sections. The separation hinges upon analyzing the computational constraints of refutation (and enforcement) via a notion of algorithmically refutable requirements (§8).

Related Work. Below, we review the most closely related work.

Our definition of refutability is inspired by Popper's notion of *testable* theories [19]. Theories of black-box testing proposed in the software engineering literature are largely concerned with the notions of test selection and test adequacy; see, e.g., [10, 27, 9, 24]. Refutable requirements have not been investigated in prior works. In contrast, *enforceable* temporal requirements have been extensively studied. Intuitively, a requirement is enforceable if there exists a reference monitor that can tell when a system violates the requirement only by observing the system's behaviors [11, 16]. Due to its technical nature, we relegate comparing refutability and enforceability to §7.

Obligations and prohibitions, as requirement types, implicitly appear in various domains of software engineering. For example, Damm and Harel introduce *existential* charts for specifying the obligatory behaviors of a system, and *universal* charts for specifying all the behaviors the system exhibits [5]. An existential chart intuitively corresponds to an obligation, and a universal chart corresponds to a *semi-monotone* requirement in our theory, which is the conjunction of an obligation and a prohibition. The notions of necessity and possibility also have a central role in modal logic. For example, Larsen and Thomsen's modal transition systems specify obligations and prohibitions through, respectively, *must* and *may* transitions [13]. Similarly, Tretmans' IOCO testing theory [24] is based on specifications that define both a lower bound and an upper bound on a system's behaviors, which roughly speaking correspond to, respectively, obligations and prohibitions (see §6). The existing works define prohibitions and obligations in concrete modeling formalisms. In contrast, we present abstract definitions which can be instantiated by the existing ones.

We briefly discuss the limitations of our theory in §9.

2 Systems and Requirements

We start with a simple abstract model of systems and requirements. A **system** is an entity that is capable of exhibiting externally observable behaviors. Operating systems, digital circuits and vending machines are all examples of systems. We keep the notion of a behavior unspecified for now. Let $\mathcal{S}$ denote the nonempty set of all systems under consideration. For example, $\mathcal{S}$ may stand for the set of all Java programs. We assume that $(\mathcal{S}, \preceq)$ is a partially-ordered set (poset), where $\preceq$ denotes a refinement relation: $S_1 \preceq S_2$ states that system S_1 refines system S_2, or that system S_2 abstracts system S_1. That is, $S_1 \preceq S_2$ means that S_1 exhibits fewer behaviors than S_2. There exists a large body of research on refinement and abstraction; see for instance [1, 14, 25]. Examples of refinement relations include trace containment and various algebraic simulation relations. In the interest of generality, we do not bind $\preceq$ to any particular relation. We write $\lceil S \rceil$ and $\lfloor S \rfloor$ respectively for the set of systems that abstract a system S and those that refine it: $\lceil S \rceil = \{S' \in \mathcal{S} \mid S \preceq S'\}$ and $\lfloor S \rfloor = \{S' \in \mathcal{S} \mid S' \preceq S\}$. We assume that the poset $(\mathcal{S}, \preceq)$ is bounded: it has a greatest element $\top$ and a least element $\bot$. The "chaos" system $\top$ (sometimes called the "weakest" system [12]), abstracts every system, and the "empty" system $\bot$ refines every system in $\mathcal{S}$. In short, our abstract model of systems is a four-tuple $(\mathcal{S}, \preceq, \bot, \top)$.

We extensionally define a **requirement** as a set of systems. A system **satisfies** a requirement R if it belongs to R. For now, we need not expound on the satisfaction relation between systems and requirements; we will give examples later. We write χ_R for a requirement R's characteristic function, which maps $\mathcal{S}$ to $\{0, 1\}$. A requirement R is **trivial** if all or none of the systems in $\mathcal{S}$ satisfy it, i.e. χ_R is a constant function. Let $\mathcal{R}$ denote the set of all requirements. It is immediate that $(\mathcal{R}, \subseteq)$, where $\subseteq$ is the standard set inclusion relation, is a complete lattice. We define the **conjunction** of two requirements R_1 and R_2, denoted $R_1 \wedge R_2$, as their meet. For a set R of systems, we write $\lceil R \rceil = \bigcup_{S \in R} \lceil S \rceil$ and $\lfloor R \rfloor = \bigcup_{S \in R} \lfloor S \rfloor$. A set R is an **upper set** if $R = \lceil R \rceil$, and a **lower set** if $R = \lfloor R \rfloor$. These terms originate from order theory.

3 Obligations and Prohibitions

A requirement is an obligation if it obliges the systems to exhibit certain (desired) behaviors, often corresponding to intended functionalities and features. For example, a requirement for a database system obliges it to provide the user with an option to commit transactions. Intuitively, this requirement cannot be violated by adding behaviors to the system, for example by providing the user the option to review transactions. The satisfaction of an obligation R is therefore abstraction-closed: $\forall S, S' \in \mathcal{S}.\ S \in R \ \wedge\ S \preceq S' \ \rightarrow\ S' \in R$. That is, an obligation is an upper set.

A requirement is a prohibition if it prohibits the systems from exhibiting certain (undesired) behaviors. For instance, consider the requirement that prohibits a database system from committing malformed transactions. Intuitively, this requirement cannot be violated by removing behaviors from the system, for example removing the option for committing transactions altogether. That is, the satisfaction of a prohibition R is refinement-closed: $\forall S, S' \in \mathcal{S}.\ S \in R \ \wedge\ S' \preceq S \ \rightarrow\ S' \in R$. In other words, a prohibition is a lower set of systems.

Definition 1. A requirement R is an **obligation** if $R = \lceil R \rceil$ and R is a **prohibition** if $R = \lfloor R \rfloor$.

The following example illustrates obligations and prohibitions.

Example 2. Consider the system model $(2^{\mathbb{N}\times\mathbb{N}}, \subseteq, \emptyset, \mathbb{N}\times\mathbb{N})$, where a system is extensionally defined as a subset of $\mathbb{N}\times\mathbb{N}$, with $\mathbb{N}$ being the set of natural numbers, and the refinement relation is the standard subset relation. For an input $i \in \mathbb{N}$, a system S can produce an output o, non-deterministically chosen from the set $\{n \in \mathbb{N} \mid (i,n) \in S\}$, and it does not produce any outputs when $\{n \in \mathbb{N} \mid (i,n) \in S\}$ is empty. We call this the **extensional input-output** system model eio.

The requirement P stipulating that systems are deterministic is a prohibition: if S is deterministic, meaning $\forall i \in \mathbb{N}.\ |\{n \in \mathbb{N} \mid (i,n) \in S\}| \leq 1$, then so is any refinement, i.e. subset, of S. In particular, the empty system satisfies the definition of determinacy.

The requirement O stipulating that systems define total relations is an obligation: if S is total, meaning $\forall i \in \mathbb{N}.\ |\{n \in \mathbb{N} \mid (i,n) \in S\}| > 0$, then so is any abstraction, i.e. superset, of S. The requirement R, stating that systems extensionally define total functions is clearly neither a prohibition nor an obligation: from $\forall i \in \mathbb{N}.\ |\{n \in \mathbb{N} \mid (i,n) \in S\}| = 1$ we cannot conclude that an arbitrary subset or superset of S defines a total function. Note that $R = P \wedge O$. ▲

A requirement R is an obligation iff χ_R is monotonically increasing in $\preceq$, that is, $S \preceq S' \rightarrow \chi_R(S) \leq \chi_R(S')$. Similarly, R is a prohibition iff χ_R is monotonically decreasing, that is, $S \preceq S' \rightarrow \chi_R(S') \leq \chi_R(S)$. Therefore, any requirement that is both an obligation and a prohibition must have a constant characteristic function. The following lemma is now immediate.

Lemma 3. If a requirement R is both an obligation and a prohibition, then R is trivial.

This lemma implies that a prohibition cannot in general be replaced with an obligation and vice versa. For example, the prohibition *smoking is forbidden* has no equivalent

obligation and the obligation *sacrifice a ram* has no equivalent prohibition. The lemma does not however imply that obligations and prohibitions exhaust the set of requirements. Namely, a **non-monotone** requirement, i.e. one whose characteristic function is neither monotonically increasing nor monotonically decreasing, is neither an obligation nor a prohibition. For instance, the requirement $R = P \wedge O$, defined in Example 2, is not monotone. Therefore, R is neither an obligation nor a prohibition.

Note that prohibitions implicitly define which behaviors are permissible. Namely, the set of permissible behaviors complements the set of prohibited ones, cf. deontic logic [26]. To avoid inconsistency, all obligatory behaviors must be permissible, but not all permissible behaviors need be obligatory. Consequently, the set of permissible behaviors for a system, delimited by the prohibitions, does not necessarily coincide with its set of obligatory behaviors.

4 Black-Box Tests

We start by defining the notion of a test setup. This notion enables us to distinguish system behaviors from what a black-box tester observes. Let $(\mathcal{S}, \preceq, \bot, \top)$ be a system model. By sampling the behaviors of a system $S \in \mathcal{S}$, a tester makes an **observation**. For now, we do not further specify observations. We give examples shortly. A **test setup** is a pair (T, α), where T is an (uninterpreted) domain of observations and α is an **order-preserving** function from $\mathcal{S}$ to 2^T, i.e., $S \preceq S' \ \rightarrow \ \alpha(S) \subseteq \alpha(S')$. Intuitively, the set $\alpha(S)$ consists of all the observations that can be made by testing a system S in this test setup. Since α is order-preserving, if t belongs to $\alpha(S)$ for some system S, then $t \in \alpha(S')$ for any system S' that abstracts S. This reflects the nature of black-box testing where analyzing a system S "by sampling" amounts to inspecting a sample of S's behaviors [6]. Therefore, if an observation can be made on S by inspecting the behaviors S exhibits, then the same observation can also be made on any system S' that abstracts S, simply because S' exhibits all of S's behaviors.

We define the function $\hat{\alpha} : T \rightarrow 2^{\mathcal{S}}$ to map an observation to the set of systems that can yield that observation. Formally, $\hat{\alpha}(t) = \{S \in \mathcal{S} \mid t \in \alpha(S)\}$, for any $t \in T$. In black-box testing, a tester knows nothing about the behaviors of the system under test beyond what is observed by interacting with it. Therefore, all the tester can conclude from an observation t is that the system under test can be *any* system that could yield t. That is, solely based on an observation t, the tester cannot distinguish between the system under test and any other member of the set $\hat{\alpha}(t)$. We call this condition the **indistinguishability condition**. Clearly black-box tests combined with, say, white-box system inspection [17], are not constrained by this condition.

The above condition delimits the knowledge a tester can obtain through black-box testing. Suppose that Ted (the tester) performs a black-box analysis of a system S. Ted cannot distinguish S from, say, $\top$, simply because $\top$ abstracts every system. This epistemic limitation is not alleviated by **exhaustive** tests: regardless of whether or not Ted samples and analyzes all the behaviors of S during testing, $\top \in \lceil S \rceil$ is still true. That is, black-box testing can neither demonstrate the absence of behaviors nor the exhaustiveness of an observation; otherwise, Ted could tell that the system under test is not $\top$, which exhibits all behaviors, thereby distinguishing S from $\top$. But, as just discussed,

this falls beyond the scope of black-box testing. The following example illustrates these points.

Example 4. Consider the eio system model and the test setup $\mathbf{T}_r = (\mathcal{S}, \lfloor\cdot\rfloor)$, where a tester may observe an arbitrary refinement of the system under test. Note that $\lfloor\cdot\rfloor$ is order-preserving and hence $\mathbf{T}_r$ is a test setup. Suppose Ted observes that the system under test S outputs 0 for input 0, and 1 for input 1. That is, Ted makes the observation $t = \{(0,0),(1,1)\}$. Ted can neither conclude that S does not output 1 for input 0, e.g. due to internal nondeterminism, nor that S extensionally defines the identity function. This is because $\top$, which abstracts t and hence belongs to $\hat{\alpha}(t)$, satisfies these requirements, and Ted cannot differentiate S from $\top$ by observing t alone. ▲

Note that the conclusions drawn above hold true regardless of whether or not observations can be carried out in a finite amount of time. We return to this point in §7.

5 Refutable Requirements

The purpose of testing a system with respect to a requirement is to refute the hypothesis that the system satisfies the requirement [19, 6, 15]. Below, we characterize the class of requirements that can be refuted using black-box tests, after presenting an illustrative special case.

Any system model $\mathsf{M} = (\mathcal{S}, \preceq, \bot, \top)$ induces a **reflexive** test setup $\mathbf{T}_r^{\mathsf{M}} = (\mathcal{S}, \lfloor\cdot\rfloor)$, where each observation on a system $S \in \mathcal{S}$ is a system in $\mathcal{S}$ that refines S. When M is clear from the context, we simply write $\mathbf{T}_r$ for M's reflexive test setup, as we did in Example 4. In the reflexive setup, testing a system S against a requirement R amounts to inspecting a refinement S_w of S to refute the hypothesis that $S \in R$. By merely observing S_w, with $S_w \in \lfloor S \rfloor$, the tester cannot distinguish S from any other system that abstracts S_w, due to the indistinguishability condition. Therefore, the tester can infer $S \notin R$ after observing S_w iff every element of $\lceil S_w \rceil$ violates R. Hence R is refutable in a reflexive test setup if, for any S that violates R, there is at least one **witness** system $S_w \in \lfloor S \rfloor$ such that any system that abstracts S_w violates R. That is, R is refutable in $\mathbf{T}_r$ if $\forall S \in \mathcal{S}.\ S \notin R \rightarrow \exists S_w \in \lfloor S \rfloor.\ \lceil S_w \rceil \cap R = \emptyset$.

Example 5. Consider a program whose input and output domains are the set of lists of natural numbers. A requirement R restricts the program's outputs to ascending lists. Suppose that a system S violates R. Then there must exist an input i for which S produces an output list o that is not ascending. Let us refer to the system that exhibits just this forbidden behavior as $S_w = \{(i, o)\}$. Clearly S_w refines S, and any system that abstracts S_w violates R by exhibiting the forbidden behavior. Therefore, R is refutable in the test setup $\mathbf{T}_r$. ▲

We now generalize the above and define refutability in an arbitrary test setup $\mathbf{T}$.

Definition 6. Let $\mathbf{T} = (T, \alpha)$ be a test setup for a system model $(\mathcal{S}, \preceq, \bot, \top)$. A requirement R is **T-refutable** if $\forall S \in \mathcal{S}.\ S \notin R \rightarrow \exists t \in \alpha(S).\ \hat{\alpha}(t) \cap R = \emptyset$.

Let R be a (T, α)-refutable requirement. Then, for any system S, $S \notin R \rightarrow \lceil S \rceil \cap R = \emptyset$, simply because α is order-preserving. The contrapositive implies that if $S_1 \in R$ and $S_2 \preceq S_1$, then $S_2 \in R$. That is, R is a prohibition. The following theorem is now immediate.

Theorem 7. *Let* $\mathbf{T}$ *be a test setup. Any* $\mathbf{T}$*-refutable requirement is a prohibition.*

Example 8. Consider the model where each system extensionally defines a binary tree where each node is colored either red or black, and $\preceq$ is the subtree relation. The requirement R stipulates that the two children of any red node must have the same color. Observing a tree t in which a red node has a red child and a black child implies that any tree that abstracts t violates R. Therefore, R is refutable in $\mathbf{T}_r$ and, due to Theorem 7, it is a prohibition. ▲

Given a system model, we say a test setup $\mathbf{T}_i$ is **more permissive** that a test setup $\mathbf{T}_j$ if any $\mathbf{T}_j$-refutable requirement is $\mathbf{T}_i$-refutable. The following lemma along with Theorem 7 imply that, in any system model M, the reflexive test setup $\mathbf{T}_r^{\mathsf{M}} = (\mathcal{S}, \lfloor \cdot \rfloor)$ is the **most permissive** test setup.

Lemma 9. In any system model M, any prohibition is $\mathbf{T}_r^{\mathsf{M}}$-refutable.

The proof is straightforward: if R is a prohibition and $S \notin R$, then $S' \notin R$ for any S' that abstracts S. Therefore, S itself can serve as the witness system demonstrating R's violation in $\mathbf{T}_r$. To further illustrate, observe that any test setup $\mathbf{T} = (T, \alpha)$ induces a set of obligations: $\mathcal{O}(\mathbf{T}) = \{\hat{\alpha}(t) \mid t \in T\}$. Testing a system S in $\mathbf{T}$ amounts to the conclusion that S satisfies an obligation that includes S, namely the obligation $\hat{\alpha}(t)$, where $t \in \alpha(S)$ is the observation obtained through testing. Therefore, the smaller $\hat{\alpha}(t)$ is, the more we learn about S by observing t; recall the indistinguishability condition. For any system S, the smallest obligation in $\mathcal{R}$ that includes S is $\lceil S \rceil$, which belongs to $\mathcal{O}(\mathbf{T}_r) = \{\lceil S \rceil \mid S \in \mathcal{S}\}$. This intuitively explains why M's reflexive setup $\mathbf{T}_r^{\mathsf{M}}$ is the most permissive test setup in any system model M. In §7, we show that $\mathbf{T}_r$ is "too permissive" in some settings, going beyond what is in practice refutable in a finite amount of time.

That $\mathbf{T}_r$ is the most permissive test setup implies that a requirement that is irrefutable in $\mathbf{T}_r$ is irrefutable for any test setup. Obligations are prominent examples of such irrefutable requirements, as stated in the following lemma, whose proof is immediate by Lemma 3 and Theorem 7.

Lemma 10. Nontrivial obligations are irrefutable in any test setup.

Example 11. Consider the setting of Example 4 and assume that the system S should satisfy the obligation O stating: systems must exhibit the behavior $(1, 0)$. Suppose Ted observes $t = \{(1, 1)\}$. Based on this, he cannot refute the hypothesis $S \in O$, simply because $\top$ abstracts t and satisfies the obligation. Note that interpreting O as the requirement P stating that *the system may output nothing but 0 for input 1* results in a refutable requirement. But O and P are not equivalent: O is an obligation and P is a prohibition; recall Lemma 3. Clearly if Ted knows that S is deterministic, then observing t would demonstrate O's violation. Determinacy itself cannot however be concluded

through black-box tests alone, simply because determinacy is a prohibition (see Example 2) and prohibitions cannot be verified through black-box tests as we prove below in Lemma 15. ▲

As the last example suggests, determinacy assumptions can play a significant role in testing. For example, passing a test that checks a program's output when the input is the empty list is in practice taken as a "proof" that the program behaves correctly on empty lists. This reasoning hinges upon the assumption that the program is deterministic.

We now turn to the irrefutability of non-monotone requirements. Suppose that a requirement R is not monotone. Although R is irrefutable by Theorem 7, it is possible that for *some* systems the violation of R can be demonstrated through tests, as explained in the following. We say a non-monotone requirement is **semi-monotone** if it is the conjunction of two monotone requirements. It is easy to prove that a requirement R is semi-monotone iff $R = \lfloor R \rfloor \wedge \lceil R \rceil$ (see Theorem 26 in Appendix A). Clearly any system S that violates the prohibition $\lfloor R \rfloor$ violates the semi-monotone R as well. Since $S \notin \lfloor R \rfloor$ can be demonstrated through tests, so can $S \notin R$. For instance, the non-monotone requirement $R = P \wedge O$, defined in Example 2, is semi-monotone. For the system $S = \{(0, n) \mid n \in \mathbb{N}\}$, any test that demonstrates $S \notin P$ also demonstrates that S violates R.

For a requirement R that is not semi-monotone, it is possible that testing can demonstrate R's violation for *none* of the systems under consideration, as the following example illustrates.

Example 12. Consider the eio model and the requirement R stating that for each $(i, o) \in S$ there exists some $(i', o) \in S$, with $i \neq i'$. This requirement, which can be seen as a simplified form of a k-anonymity requirement [23], intuitively states that by solely inspecting a system's outputs, an observer cannot determine whether or not the input is some particular $i \in \mathbb{N}$. Note that R is not monotone. Moreover, R's violation (for any system) cannot be demonstrated through tests in any test setup (T, α): every observation $t \in T$ obtained by testing any system belongs to $\alpha(\top)$, and $\top \in R$. It is easy to check that $\lceil R \rceil \wedge \lfloor R \rfloor = \mathcal{S}$ and hence R is not semi-monotone. ▲

6 Verifiable Requirements

We define testing with the purpose of verifying the satisfaction of a requirement as dual to testing for refutation.

Definition 13. Let $\mathbf{T} = (T, \alpha)$ be a test setup for a system model $(\mathcal{S}, \preceq, \bot, \top)$. A requirement R is **T-verifiable** if $\forall S \in \mathcal{S}.\ S \in R \rightarrow \exists t \in \alpha(S).\ \hat{\alpha}(t) \subseteq R$.

In particular, a requirement R is $\mathbf{T}_r^{\mathsf{M}}$-verifiable in the system model $\mathsf{M} = (\mathcal{S}, \preceq, \bot, \top)$ if $\forall S \in \mathcal{S}.\ S \in R \rightarrow \exists S_w \in \lfloor S \rfloor.\ \lceil S_w \rceil \subseteq R$. That is, if there exists a witness system S_w that refines S and any system that abstracts S_w satisfies R, then by observing S_w we have conclusively demonstrated $S \in R$. The following theorem is dual to Theorem 7. Its proof is immediate.

Theorem 14. *Let* $\mathbf{T}$ *be a test setup. Any* $\mathbf{T}$*-verifiable requirement is an obligation.*

An observation $t \in \alpha(S)$ shows that the system S satisfies the obligation $O = \hat{\alpha}(t)$. It also proves that $S \in R$ for any requirement $R \supseteq O$. Therefore, as O becomes smaller, more obligations are proved by the observation. This also explains why $\mathbf{T}_r$ is the most permissive test setup for verification: any $\mathbf{T}$-verifiable requirement is $\mathbf{T}_r$-verifiable. Consequently, a requirement that is not $\mathbf{T}_r$-verifiable is non-verifiable in any test setup. Prohibitions are prominent examples of such non-verifiable requirements. The following lemma's proof is straightforward.

Lemma 15. Nontrivial prohibitions are non-verifiable in any test setup.

The lemma expresses the essence of Dijkstra's often-quoted statement that "program testing can be used to show the presence of bugs, but never to show their absence" [6]. Contrary to the folklore, this does not mean that no requirement is verifiable through black-box tests. For instance, the requirement that obliges a magic 8-ball to output *ask again later* is clearly verifiable through black-box tests: observing this output once demonstrates the obligation's satisfaction. The following example illustrates this point. We return to this example in §7 where we investigate temporal requirements.

Example 16. Consider the setting where a system S is identified with the set $b(S)$ of its behaviors, and $S_1 \preceq S_2$ denotes $b(S_1) \subseteq b(S_2)$. Suppose that a system behavior is a sequence of events and e is an event. Assume that a system S satisfies the requirement R_e stating that systems exhibit at least one behavior where e eventually appears. Note that R_e is an obligation since its satisfaction is abstraction-closed. Now, observing a refinement S_w of S where S_w exhibits one behavior π in which e eventually appears demonstrates $S \in R_e$: any abstraction of S_w exhibits π as well, hence satisfying R_e. We conclude that the obligation R_e is verifiable through tests in $\mathbf{T}_r$. ▲

We can now sharpen Dijkstra's dictum to: **(D)** *Program testing can be used to show the presence of behaviors, but never to show their absence*. If a software *bug* is a prohibited behavior, then **(D)** coincides with Dijkstra's statement, simply stipulating that prohibitions are refutable, but not verifiable. However, if a bug is the absence of an obliged behavior, then **(D)** translates to: program testing can be used to show the absence of bugs, but never to show their presence. This statement, which is dual to Dijkstra's, simply stipulates that obligations are verifiable, but not refutable.

We conclude this section with an intuitive interpretation of refutability and verifiability. The examples thus far given in the paper suggest that a system S satisfies an obligation O if the set of desired behaviors that O obliges is included in the set of behaviors of S. Any violation of O is therefore due to the behaviors that S "lacks". Consequently, O can be seen as a "lower-bound" for the set of S's behaviors. Similarly, S satisfies a prohibition P iff the set of behaviors of S is contained in the set of behaviors P permits. Any violation of P is therefore due to "excessive" behaviors of S. In this sense, P constitutes an "upper-bound" for the set of S's behaviors; see Figure 1.

7 Refutation in Finite Time

A requirement that is deemed refutable in our theory might not be refutable in practice. For example, a requirement whose refutation hinges upon measuring the exact momentum and position of a quantum object is impossible to refute due to the laws of physics.

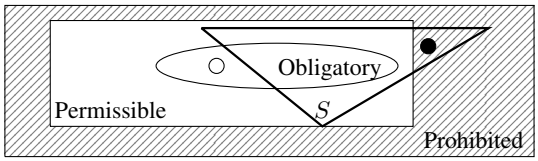

Fig. 1. The hatched area stands for the set of prohibited behaviors. The white box is the set of permissible ones which includes the set of obligatory behaviors, represented by the oval. The triangle represents a system S's behaviors. The white circle represents a violation of the obligation denoted by the oval, and the black circle represents a violation of the prohibition depicted by the hatched area.

This limitation, not unexpectedly, does not follow from our logical theory of tests and refutation. Below, we extend our theory to account for a practically relevant limitation of system testing: we consider refutation through black-box tests that proceed in a finite amount of time.

In a system model $(\mathcal{S}, \preceq, \bot, \top)$, to show that a requirement R's satisfaction is refutable through tests in a finite amount of time, we prove that R is $\mathbf{T}$-refutable in a setup $\mathbf{T} = (T, \alpha)$ where (1) every observation in $\bigcup_{S \in \mathcal{S}} \alpha(S)$ consists of finitely many elements and (2) each such element is observable in finite time. In this case, we say R is **finitely refutable** in $\mathbf{T}$. The notion of **finite verifiability** is defined dually.

Condition (2) above refers to the world: determining whether a given element can be observed in finite time falls outside our theory's scope, and this condition's satisfaction must be substantiated by other means. Thus our theory cannot establish a requirement's finite (ir)refutability unless assumptions are made about what can be observed in finite time in the world. The following example illustrates this point.

Example 17. Consider the eio system model and the family $\mathbf{T}_k = ((\mathbb{N} \times \mathbb{N})^k, \alpha_k)$ of test setups, where $k \geq 1$ and α_k maps any system S to S^k, inductively defined as $S^1 = S$ and $S^{k+1} = S \times S^k$. Testing a system S in the setup $\mathbf{T}_k$ amounts to observing k input-output pairs belonging to S. Assume that natural numbers are observable in finite time. Then, observing every element of $(\mathbb{N} \times \mathbb{N})^k$, where $k \geq 1$ belongs to $\mathbb{N}$, takes finite time. The requirement R_{nz} stating that *systems never output zero* is, under this assumption, finitely refutable in $\mathbf{T}_1$. Now consider the requirement R_{fz}, stating that *systems may output zero for at most finitely many inputs*. It is easy to check that R_{fz}, although refutable in the reflexive test setup $\mathbf{T}_r$, is not $\mathbf{T}_k$-refutable for any $k \geq 1$.

It now seems reasonable to conclude that R_{fz} is not finitely refutable: no finite set of behaviors can refute R_{fz}. This conclusion does not however follow from our theory. To illustrate, consider an alternative test setup $\mathbf{T} = (\{*, \omega\}, \alpha)$, where $\alpha(S) = \{*\}$ if S outputs zero for finitely many inputs, and $\alpha(S) = \{*, \omega\}$ otherwise. Since α is order-preserving, $\mathbf{T}$ is formally a test setup. The requirement R_{fz} is finitely refutable in $\mathbf{T}$, under the assumption that the elements of $\{*, \omega\}$ are observable in finite time, which is the essence of Condition (2) above. Whether this is a tenable assumption cannot be settled inside our theory. Although $\mathbf{T}$ hardly appears realizable, such observations

are possible in certain cases, for example by measuring the electromagnetic radiation emitted from a black-box system, cf. [22]. ▲

Condition (1) above is satisfied if finitely many behaviors of the system under test are sampled for each observation, and only a finite portion of those behaviors are inspected even when the behaviors themselves are not finite objects. We illustrate this point with an example.

Example 18. Consider the system model $(2^{\mathbb{R}\times\mathbb{R}}, \subseteq, \emptyset, \mathbb{R}\times\mathbb{R})$, where $\mathbb{R}$ is the set of real numbers. This system model is similar to the eio model except its input-output pairs belong to $\mathbb{R}$. Define $pre(r)$ as the set of finite truncations of the decimal expansion of a real number r. For instance, $pre(\sqrt{2}) = \{1, 1.4, 1.41, 1.414, 1.4142, \cdots\}$. Note that a real number can have more than one decimal expansions, for example, 1 and $0.999\cdots$, but accounting for this point is unnecessary for our discussion here. We define the test setup $\mathbf{T} = (\mathbb{F}\times\mathbb{F}, \alpha)$, where $\mathbb{F}$ is the set of rational numbers that have a finite decimal expansion and α maps any system S to the set $\bigcup_{(i,o)\in S} pre(i)\times pre(o)$. An observation of a system S in this setup is a pair (f_1, f_2), where f_1 is a truncation of an input i and f_2 is a truncation of an output o, where $(i, o) \in S$. That is, we may observe only finite portions of the decimal expansions of the inputs and outputs. Assume that $\mathbb{F}$'s elements are observable in finite time. That is, any $\mathbf{T}$-refutable requirement is finitely refutable.

Now consider the requirement $R_<$, stating that system outputs are strictly smaller than $\sqrt{2}$. Clearly $R_<$ is a prohibition, hence $\mathbf{T}_r$-refutable. Below, we show that $R_<$ is not $\mathbf{T}$-refutable. Define the system $S = \{(1, 1.4142\cdots)\}$, which outputs $\sqrt{2}$, decimally expanded, for the input 1. Even though S violates $R_<$, no truncation of S's output's decimal expansion conclusively demonstrates this, because the set of permissible outputs according to $R_<$, namely $\{o \in \mathbb{R} \mid o < \sqrt{2}\}$, is not a closed set in $\mathbb{R}$'s standard topology. That is, there is a number, namely $\sqrt{2}$, that is arbitrarily close to this set, but is not a member of the set. No finite truncation of this numbers decimal expansion can therefore conclusively determine whether it is a member, or not. We therefore conclude that $R_<$ is not $\mathbf{T}$-refutable. An analogous argument shows that the requirement $R_\leq$ stating that system outputs must be less than or equal to $\sqrt{2}$ is $\mathbf{T}$-refutable, and hence finitely refutable, because the set of permissible outputs it induces, namely $(-\infty, \sqrt{2}]$, is topologically closed. ▲

The example suggests that there is a fundamental connection between refutability and topological closure when system behaviors are infinite sequences. This connection has been investigated by Alpern and Schneider in the context of temporal properties [2], which we turn to next.

To investigate temporal requirements, we model systems that induce infinitely long sequences of events, such as operating systems, and their requirements following [4]. Let Σ be an alphabet (e.g. of events or states), where every element of Σ can be observed in finite time. We write Σ^* and Σ^ω for, respectively, the sets of finite and countably infinite sequences of Σ's elements. A behavior is an element of Σ^ω and a system is a set of behaviors. The complete lattice $(2^{\Sigma^\omega}, \subseteq, \emptyset, \Sigma^\omega)$ instantiates our system model, defined in §2. For a behavior $\pi \in \Sigma^\omega$, we write $pre(\pi)$ for the set of all finite prefixes of π, and we denote the concatenation of an element of Σ^* with one of Σ^ω by their juxtaposition. As usual, a requirement is a set of systems.

We define the test setup $\mathbf{T}_*$ as (T_*, α_*), where T_* is the set of all finite subsets of Σ^*, and $\alpha_*(S)$ is the set of all finite subsets of $\bigcup_{\pi \in S} pre(\pi)$ for a system S. Intuitively, any element of $\alpha_*(S)$ is a possible observation of S where finite prefixes of finitely many behaviors of S are observed. For any $\mathbf{T}_*$-refutable requirement R and any $S \notin R$, there exists a finite (witness) set t_w of finite prefixes of S's behaviors such that any system S' that could have yielded the observation t_w, i.e. $t_w \in \alpha_*(S')$, violates R. Clearly, every $\mathbf{T}_*$-refutable requirement is finitely refutable. Next, we relate $\mathbf{T}_*$-refutability and $\mathbf{T}_*$-verifiability to the notions of *properties* and *hyper-properties*.

A **property** is a set of (permitted) behaviors [18, 2], i.e. a subset of Σ^ω. We have (extensionally) defined a system as a set of behaviors. A property can therefore be seen as a system. By overloading the notion of satisfaction, we say a system S *satisfies* a property ϕ if $S \subseteq \phi$. Any property ϕ thereby defines a refinement-closed requirement $R_\phi = \lfloor \phi \rfloor$. A property ϕ is **safety** if $\forall \pi \notin \phi. \exists \sigma \in pre(\pi). \forall \pi' \in \Sigma^\omega. \sigma\pi' \notin \phi$ and **liveness** if $\forall \sigma \in \Sigma^*. \exists \pi \in \Sigma^\omega. \sigma\pi \in \phi$. That is, safety and liveness properties are closed and dense sets, respectively [2]. The following example illustrates properties.

Example 19. A linear-time temporal logic (LTL) formula ϕ defines a set $b(\phi)$ of behaviors or executions; see, e.g., [18]. A system S satisfies the requirement expressed by ϕ if the set of system behaviors is contained in $b(\phi)$. Clearly if S satisfies the requirement and $S' \subseteq S$, for some system S', then S' satisfies it as well. Here the satisfaction relation is refinement-closed, with $S_1 \preceq S_2$ if $S_1 \subseteq S_2$. Hence, for every ϕ, regardless of whether $b(\phi)$ is a safety property, a liveness property, or an intersection of the two [2], the requirement expressed by ϕ is a prohibition. Note that the obligation R_e of Example 16 is not expressible in LTL, where the satisfaction relation is refinement-closed as just discussed, because then R_e would have to be trivial by Lemma 3. However, R_e is expressed as $\mathsf{EF}\ e$ in the computation tree logic CTL. This argument amounts to a simple proof for the well-known result that LTL is *not* more expressive than CTL [7]. ▲

Theorem 20. *A temporal property ϕ is $\mathbf{T}_*$-refutable iff ϕ is safety. Moreover, all temporal properties are $\mathbf{T}_r$-refutable and any $\mathbf{T}_*$-verifiable property is trivial.*

The theorem, whose proof hinges upon Lemma 3 and Theorem 7, implies that nontrivial liveness properties, although $\mathbf{T}_r$-refutable, are not $\mathbf{T}_*$-refutable; cf. [8]. We now turn to hyper-properties.

A **hyper-property** is a set of properties [4], i.e. a requirement in our model. A system S satisfies a hyper-property $\mathbb{H}$, if $S \in \mathbb{H}$. A hyper-property $\mathbb{H}$ is **hyper-safety** if for any $S \notin \mathbb{H}$, there exists an observation $t \in \alpha(S)$ such that $\forall S' \in \hat{\alpha}(t).\ S' \notin \mathbb{H}$; see [4]. It is easy to check that a temporal requirement R is hyper-safety iff R is $\mathbf{T}_*$-refutable. Now it is immediate by Theorem 7 that any hyper-safety requirement is a prohibition. Therefore, finitely verifiable hyper-safety requirements must be trivial. These results show how existing, specialized concepts and their refutability follow as special cases of the notions we defined. For instance, Example 16's requirement R_e, which is clearly finitely verifiable in $\mathbf{T}_*$, cannot be hyper-safety due to the above results and it is therefore not finitely refutable in $\mathbf{T}_*$.

We conclude this section with another application of our theory and consider refutation through system self-composition. Suppose we want to refute the hypothesis H

stating that a plane figure S is a circle by observing a number of points lying on the figure. Since any three (non-collinear) points define a circle, in general we must observe at least four points lying on the figure to refute H. Alternatively, we may consider the fictitious entity defined by the relation r^4, where the relation r is obtained from S: $(x, y) \in r$ iff the point (x, y) lies on S. Then, observing a single element of the set r^4, which consists of four-tuples of the coordinates of the figure's points, can refute H. This line of reasoning is central to the **self composition** technique [3]: to refute a k-*property*, which is a temporal property that cannot be refuted by observing less than k system behaviors [4], one can make a single observation on the fictitious entity that contains k copies of the system under test. Returning now to the family of test setups $\mathbf{T}_k$, with $k \geq 1$, defined in Example 17, we remark that a requirement that is $\mathbf{T}_{k+1}$-refutable, but not $\mathbf{T}_k$-refutable, can be refuted by observing a single behavior of the entity that is found by self-composing the system under test no less than $k + 1$ times. We proceed with an example.

Example 21. The prohibition P, stating that systems are deterministic, given in Example 2, is not $\mathbf{T}_1$-refutable: observing any single behavior of a non-deterministic eio system S is insufficient to refute the hypothesis $S \in P$. However, observing a single element of S^2 can demonstrate P's violation. Therefore, P is $\mathbf{T}_2$-refutable: determinacy can be refuted by self-composing the system under test. ▲

Intuitively, the less stringently a requirement is specified, the larger is the number of times the system under test must be self-composed to facilitate the requirement's refutability. We illustrate this point with a simple example. Consider the eio system model and the prohibition R defined as: S satisfies R iff $\forall(i, o) \in S.\ o = f(i)$, where f is a function from $\mathbb{N}$ to $\mathbb{N}$. Suppose that f is not available, but it is known that f is monotonically increasing: $i_1 \leq i_2$ implies $f(i_1) \leq f(i_2)$. We write R' for the corresponding (less stringent) requirement: $S \in R'$ iff $\forall(i_1, o_1), (i_2, o_2) \in S$, $i_1 \leq i_2$ implies $o_1 \leq o_2$. The requirement R is $\mathbf{T}_1$-refutable, whereas R', which is a superset of R, is not: R''s test oracle cannot make a decision solely based on a single input-output pair; a **test oracle** for a requirement R is a (partial) decision function that given a set of system behaviors tells whether the system violates R. Note however that R' is $\mathbf{T}_2$-refutable. Since $\mathbf{T}_{k+1}$ is strictly more permissive than $\mathbf{T}_k$, we conclude that the less stringently specified R is (in the above sense), the more a system must be sampled, or self-composed, to facilitate a test oracle for R; cf. [21]. That is, a partially specified requirement is harder to refute, because it leaves more leeway.

8 Algorithmic Refutability

We now characterize the requirements whose violation can be demonstrated through algorithmic means. We start with an auxiliary definition. Any requirement R induces a set Ω_R of **irremediable observations** $\{t \in T \mid \hat{\alpha}(t) \cap R = \emptyset\}$ in a test setup $\mathbf{T} = (T, \alpha)$. It follows that a system S violates a $\mathbf{T}$-refutable R iff $\alpha(S) \cap \Omega_R \neq \emptyset$. Intuitively, a requirement is algorithmically refutable only if it induces a recursively enumerable set of irremediable observations. Recall that, given a countable set U, a

set $E \subseteq U$ is **recursively enumerable** if there is a (semi-)algorithm $\mathcal{A}_E$ that terminates and outputs *true* for any input $u \in U$ that is a member of E. If $u \notin E$, then $\mathcal{A}_E$ does not terminate.

Definition 22. A requirement R is **algorithmically refutable** in the test setup $\mathbf{T} = (T, \alpha)$ if R is finitely refutable in $\mathbf{T}$, and Ω_R is a recursively enumerable subset of the countable set T.

If a system S violates an algorithmically refutable requirement R in $\mathbf{T}$, then there is at least one observation $t \in \alpha(S)$ that can be carried out in finite time, where $\mathcal{A}_{\Omega_R}$ terminates on t and outputs *true*. Here, *true* means $t \in \Omega_R$, demonstrating $S \notin R$. Observing such a t through testing, therefore, conclusively refutes the hypothesis $S \in R$. The following example illustrates Definition 22.

Example 23. Consider the prohibition P for eio systems stating that a system may never output 0 for an odd input. Clearly, P is $\mathbf{T}_1$-refutable, and whether an observation $\{(i, o)\}$ is irremediable is decidable since the set $\Omega_P = \{\{(2i + 1, 0)\} \in T_1 \mid i \in \mathbb{N}\}$ is recursive. If natural numbers are finitely observable, then P is algorithmically refutable in $\mathbf{T}_1$: any S that violates P has a finitely observable behavior, e.g. $t = \{(3, 0)\}$, and whether, or not, $t \in \Omega_P$ can be decided by a Turing machine. ▲

Let R be an algorithmically refutable requirement in $\mathbf{T} = (T, \alpha)$, and S be a system. The decision problem that asks whether S violates R is semi-decidable, if $\alpha(S)$ is a recursively enumerable subset of T. We illustrate this point using the following **test algorithm**, which relies on *dovetailing*. For a formal treatment of dovetailing, which is a poor man's parallelization technique, see, e.g., [20].

Algorithm 24. Fix an arbitrary total order on T's elements. Dovetail $\mathcal{A}_{\alpha(S)}$'s computations on the elements of T. In parallel, dovetail $\mathcal{A}_{\Omega_R}$'s computations on those observations for which $\mathcal{A}_{\alpha(S)}$ terminates. Output *true* and terminate, when any computation of $\mathcal{A}_{\Omega_R}$ terminates.

If $S \notin R$, then there exists at least one observation t_w in the set $\alpha(S) \cap \Omega_R$. The test algorithm is bound to terminate on t_w and output *true*, thus demonstrating $S \notin R$ in finite time. However, if $S \in R$, then the test (semi-)algorithm does not terminate. Ideally, standard test selection methods [15] place likely witnesses of R's violation early in the ordering assumed on T. Note that the above test algorithm achieves the (impractical) ideal of testing: it not only has "a high probability of detecting an as yet undiscovered error" [15], the algorithm is in fact guaranteed to reveal flaws in any system that violates an algorithmically refutable requirement. We proceed with an example.

Example 25. Fix an ordering on the set of all Turing machines: $M_0, M_1, \cdots$. In the eio model the requirement R is defined as: $S \in R$ if for any $(i, o) \in S$ the machine M_i diverges on o. It is easy to check that R is $\mathbf{T}_1$-refutable and the set Ω_R is recursively enumerable. Suppose that $\alpha_1(S)$ is recursively enumerable for a system S. That is, $\mathcal{A}_{\alpha(S)}$ is guaranteed to terminate on any $(i, o) \in S$ in the universe $\mathbb{N} \times \mathbb{N}$. If $S \notin R$, then there is a "witness" $(i_w, o_w) \in S$ where M_{i_w} terminates on o_w. Therefore, dovetailing M_i's computations on o for all (i, o) on which $\mathcal{A}_{\alpha(S)}$ terminates is bound to witness a terminating computation, thus demonstrating $S \notin R$ in finite time. ▲

Next, we apply the notion of algorithmic refutability to the temporal requirements introduced in §7 and the corresponding test setup $\mathbf{T}_*$. It is easy to check that a safety property ϕ is algorithmically refutable iff ϕ's set of *irremediable sequences* $\nabla_\phi = \{\sigma \in \Sigma^* \mid \forall \pi \in \Sigma^\omega.\ \sigma\pi \notin \phi\}$ is recursively enumerable. This condition separates refutability from **enforceability**, as explained below. To enforce the safety property ϕ on a system S, a reference monitor observes some $t \in \alpha(S)$. If t demonstrates that S violates ϕ, then the monitor *stops* S. Otherwise, the monitor *permits* S to continue its execution. For enforcement, the set ∇_ϕ must therefore be recursive [11]. It then follows that any enforceable temporal property is algorithmically refutable. An algorithmically refutable property need not however be enforceable: any property ϕ, where ∇_ϕ is recursively enumerable but not recursive, is algorithmically refutable, but not enforceable.

To further illustrate the relationship between refutability and enforceability, we define **weak enforceability** for a hyper-safety requirement R as follows. By monitoring the executions of a system S, a monitor observes some $t \in \alpha_*(S)$. If t does *not* conclusively demonstrate $S \notin R$, then the monitor permits S to continue. However, if t does conclusively demonstrate R's violation, then the monitor may either stop S, or diverge and thereby stall S. Recall that a system S violates a hyper-safety requirement R iff $\alpha_*(S) \cap \Omega_R \neq \emptyset$. To weakly enforce R, the set Ω_R must therefore be co-recursively enumerable, i.e. $T_* \setminus \Omega_R$ must be recursively enumerable. This observation, which concurs with [16, Theorem 4.2], illustrates that weak enforceability and algorithmic refutability are complementary in the sense that the former requires Ω_R to be co-recursively enumerable and the latter requires Ω_R to be recursively enumerable. This duality between refutability and enforceability becomes evident only after explicating the computational constraints of testing and enforcement.

9 Concluding Remarks

We have formally characterized the classes of refutable and verifiable requirements for black-box tests. Naturally black-box testing can be combined with other analysis techniques, like white-box system inspection; see, e.g., [17]. The indistinguishability condition of §4, stating that the system under test can be any abstraction of an observation obtained through tests, would then no longer be applicable. For instance, if the system under test is known to be deterministic, then clearly more requirements become refutable as discussed in §5. In other words, augmenting black-box analysis with knowledge that itself cannot be verified through black-box tests (e.g., coming from white-box analysis) would expand the analysis's capabilities, leading to more powerful refutation methods. Developing such an extension of our theory and exploring its applications remain as future work.

We remark that our theory of tests and refutation is not readily applicable to probabilistic constraints. For example, a gambling regulation requiring that slot machines have a 75% payout cannot be refuted through black-box test. Nevertheless, tests refuting such probabilistic constraints with a controllable margin of error can be devised. Developing a corresponding theory of tests and refutation also remains as future work.

Acknowledgments. We thank E. Fang, M. Guarnieri, G. Petric Maretic, S. Radomirovic, C. Sprenger, and E. Zalinescu for their comments on the paper.

References

1. Martin Abadi and Leslie Lamport. The existence of refinement mappings. In *LICS*, pages 165–175. IEEE, 1988.
2. Bowen Alpern and Fred Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985.
3. Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *CSFW '04*, pages 100–114. IEEE Computer Society, 2004.
4. Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
5. Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
6. Edsger W. Dijkstra. Notes on structured programming. Technical Report T.H. Report 70-WSK-03, Technological University Eindhoven, April 1970.
7. E. Allen Emerson and Joseph Halpern. "Sometimes" and "Not Never" Revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.
8. Yliès Falcone, Jean-Claude Fernandez, Thierry Jéron, Hervé Marchand, and Laurent Mounier. More testable properties. *STTT*, 14(4):407–437, 2012.
9. Marie-Claude Gaudel. Testing can be formal, too. In *TAPSOFT '95*, pages 82–96. Springer, 1995.
10. John Goodenough and Susan Gerhart. Toward a theory of test data selection. *IEEE Trans. Softw. Eng.*, 1(2):156–173, 1975.
11. Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.*, 28(1):175–205, 2006.
12. C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.
13. Kim Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE, 1988.
14. Carroll Morgan. *Programming from Specifications*. Prentice Hall, 1998.
15. Glenford Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley, 2011.
16. Minh Ngo, Fabio Massacci, Dimiter Milushev, and Frank Piessens. Runtime enforcement of security policies on black box reactive programs. In *POPL '15*, pages 43–54. ACM, 2015.
17. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 2005.
18. Amir Pnueli. The temporal logic of programs. In *FOCS '77*, pages 46–57. IEEE, 1977.
19. Karl Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge, 1963.
20. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.
21. Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, 2016.
22. François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT '09*, pages 443–461. Springer-Verlag, 2009.
23. Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
24. Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 1–38. Springer, 2008.
25. Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR '90*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.
26. Georg H. von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.
27. Elaine J. Weyuker. Axiomatizing software test data adequacy. *IEEE Trans. Softw. Eng.*, 12(12):1128–1138, 1986.

A Proofs

We first present the proofs of the lemmas and theorems that are given in the paper. Afterwards, we formally state and prove the claim that a requirement is semi-monotone iff it is the intersection of its upper set and its lower set, which is mentioned in §5.

Proof (Lemma 3's Proof). If no system satisfies R, then R is trivial. Suppose that some system S satisfies R. Then, every system in $\lfloor\lceil S\rceil\rfloor$ satisfies R, because R is an obligation and a prohibition. As $\lfloor\lceil S\rceil\rfloor = \mathcal{S}$, for any $S \in \mathcal{S}$, we conclude that every system satisfies R. That is, R is trivial.

Proof (Theorem 7's Proof). Suppose R is $\mathbf{T}$-refutable, with $\mathbf{T} = (T, \alpha)$. We prove that R is a prohibition. If R is empty, then R is a trivial prohibition. Assume that R is nonempty and let $S \in R$. Now, suppose $S' \preceq S$. All we need to prove is that $S' \in R$. We present a proof by contradiction.

Assume that $S' \notin R$. Then $\exists t \in T.\ \hat{\alpha}(t) \cap R = \emptyset$ simply because R is $\mathbf{T}$-refutable. Since α is order-preserving and $S' \preceq S$, we have $t \in \alpha(S)$. Therefore, $S \in \hat{\alpha}(t)$. This entails $S \notin R$, which contradicts the assumption $S \in R$. We conclude that $S' \in R$. Therefore, R is a prohibition.

Proof (Lemma 9's Proof). Fix a system model $\mathsf{M} = (\mathcal{S}, \preceq, \bot, \top)$, and assume that R is prohibition. We show that R is $\mathbf{T}_r^{\mathsf{M}}$-refutable, where $\mathbf{T}_r^{\mathsf{M}} = (\mathcal{S}, \lfloor\cdot\rfloor)$. Assume that some system S violates R. Since R is a prohibition, any system that abstracts S violates R. Moreover, $S \in \lfloor S\rfloor$. We conclude that $\exists S_w \in \lfloor S\rfloor.\ \lceil S_w\rceil \cap R = \emptyset$, namely $S_w = S$. Therefore, R is $\mathbf{T}_r^{\mathsf{M}}$-refutable.

Proof (Lemma 10's Proof). Suppose R is a nontrivial obligation. We prove by contradiction that R is not refutable in any test setup.

Assume that R is $\mathbf{T}$-refutable in some test setup $\mathbf{T}$. By Theorem 7, R is a prohibition. Then, R must be trivial by Lemma 3, because R is both a prohibition and an obligation. That R is a trivial contradicts the assumption that R is a nontrivial obligation. We conclude that R is not refutable in any test setup.

Proof (Theorem 14's Proof). Suppose R is $\mathbf{T}$-verifiable, with $\mathbf{T} = (T, \alpha)$. We prove that R is an obligation. If R is empty, then R is a trivial obligation. Assume that R is nonempty and let $S \in R$. Now, suppose $S \preceq S'$. All we need to prove is that $S' \in R$. Since R is $\mathbf{T}$-verifiable, from $S \in R$ we conclude $\exists t \in \alpha(S).\ \hat{\alpha}(S) \subseteq R$. As α is order-preserving and $S \preceq S'$, we have $t \in \alpha(S')$. That is, $S' \in \hat{\alpha}(S)$. We conclude that $S' \in R$. Therefore, R is an obligation.

Proof (Lemma 15's Proof). Suppose R is a nontrivial prohibition. We prove by contradiction that R is not verifiable in any test setup.

Assume that R is $\mathbf{T}$-verifiable in some test setup $\mathbf{T}$. By Theorem 14, R is an obligation. Then, R must be trivial by Lemma 3, because R is both a prohibition and an obligation. That R is a trivial contradicts the assumption that R is a nontrivial prohibition. We conclude that R is not verifiable in any test setup.

Proof (Theorem 20's Proof). We split the proof into several parts.

(1) Suppose that a property ϕ is $\mathbf{T}_*$-refutable. We show that ϕ is safety. Assume that $\pi \notin \phi$, for some $\pi \in \Sigma^\omega$. Then, the system $S_\pi = \{\pi\}$ violates ϕ. Now, by ϕ's $\mathbf{T}_*$-refutability, there exists a finite set t of ϕ's finite prefixes that demonstrates $S_\pi \notin R_\phi$, where $R_\phi = \lfloor \phi \rfloor$. Let σ be the longest element in t; note that since $\{\pi\}$ is a singleton, there always exists a single longest element in t. Then, for any $\pi' \in \Sigma^\omega$, the system $S_{\pi'} = \{\sigma\pi'\}$ violates ϕ, simply because t belongs to $\alpha(S_{\pi'})$. We conclude that $\sigma\pi' \notin \phi$, for all $\pi' \in \Sigma^\omega$. That is, ϕ is a safety temporal property.

(2) Suppose that ϕ is safety. We show that ϕ is $\mathbf{T}_*$-refutable. Assume that a system S violates ϕ. That is, $\exists \pi \in S.\ \pi \notin \phi$. Since ϕ is safety, a finite prefix of π, say σ, satisfies the following condition: $\forall \pi' \in \Sigma^\omega.\ \sigma\pi' \notin \phi$. Now, define the observation $t \in T_*$ as $\{\sigma\}$. Note that $t \in \alpha(S)$, and moreover $\hat{\alpha}(t) \cap R_\phi = \emptyset$ due to the above condition. This shows that ϕ is $\mathbf{T}_*$-refutable.

(3) Any temporal property ϕ is $\mathbf{T}_r$-refutable because R_ϕ's satisfaction is abstraction-closed, for any ϕ. Then, by Lemmas 3 and 15, any $\mathbf{T}_r$-verifiable or $\mathbf{T}_*$-verifiable property must be trivial.

Theorem 26. *A requirement R is semi-monotone iff $R = \lfloor R \rfloor \wedge \lceil R \rceil$.*

Proof. We split the proof into two parts, reflecting the theorem's two statements.

(1) Assume that R is semi-monotone. We show that $R = \lfloor R \rfloor \wedge \lceil R \rceil$. Clearly $R \subseteq \lceil R \rceil \wedge \lfloor R \rfloor$, for any requirement R. All we need to prove is that $\lceil R \rceil \wedge \lfloor R \rfloor \subseteq R$. If $\lceil R \rceil \wedge \lfloor R \rfloor = \emptyset$, then the claim trivially holds. Suppose $S \in \lceil R \rceil \wedge \lfloor R \rfloor$ for some system S. From $S \in \lceil R \rceil$, we conclude $\exists S_- \in R.\ S_- \preceq S$. Similarly, from $S \in \lfloor R \rfloor$, we conclude $\exists S_+ \in R.\ S \preceq S_+$. In short, we have $S_- \preceq S \preceq S_+$, $S_- \in R$, and $S_+ \in R$. Then, Lemma 27 below implies that $S \in R$, simply because R is semi-monotone. Therefore, if R is semi-monotone, then $R = \lceil R \rceil \wedge \lfloor R \rfloor$.

(2) Assume that $R = \lfloor R \rfloor \wedge \lceil R \rceil$. We prove that R is semi-monotone. Note that for any requirement Q, the requirement $\lfloor Q \rfloor$ is a prohibition, hence monotone. Moreover, $\lceil Q \rceil$ is an obligation, hence monotone. Therefore, $\lfloor Q \rfloor \wedge \lceil Q \rceil$ is semi-monotone, that is the intersection of two monotone requirements, for any requirement Q. In particular, R is semi-monotone because $R = \lfloor R \rfloor \wedge \lceil R \rceil$.

Lemma 27. If R is a semi-monotone requirement, then for any three systems S_-, S, and S_+ the following condition holds.

$$S_- \preceq S \preceq S_+ \ \wedge\ S_- \in R \ \wedge\ S_+ \in R \ \rightarrow\ S \in R$$

Proof. Either (1) R is monotone, that is R is the conjunction of two prohibitions or the conjunction of two obligations, or (2) R is the conjunction of a prohibition P and an obligation O. The lemma's claim is immediate for case (1). Let us consider case (2). Suppose $S_- \preceq S \preceq S_+ \ \wedge\ S_- \in R \ \wedge\ S_+ \in R$. Note that $S_- \in R$ implies that $S_- \in O$. Then, $S_- \preceq S$ implies that $S \in O$. Similarly, $S_+ \in R$ implies that $S_+ \in P$. Then, $S \preceq S_+$ implies that $S \in P$. These two statements show that $S \in P \wedge O$. That is, $S \in R$.