Abstract

In this paper we take an abstract set based approach to testing. With this approach we are able to discuss testing issues which are totally representation free. We develop a game theoretic approach to testing and obtain some complexity results from this approach. We develop a notion of testing in the limit and discuss alternative definitions of testing.

Introduction

Over a number of years we have been investigating foundational issues concerning a simple model of testing [5], [4], and [3]. In its most abstract setting this model of testing necessarily subsumes all other testing models. This subsumption does not imply that the model is more useful than existing models (in and of itself it is not); it does yield insights into the elements common to all testing models and can be used as a framework to unify and compare such models.

In this paper we continue our foundational studies of testing. We deemphasize the connections between our testing theory and inductive inference (though they still exist) constructing our set based theory without reference to inductive inference. We introduce some new complexity results for testing simple classes of programs by developing a game theoretic approach to testing. From this we obtain both feasibility and infeasibility results. Finally, we develop a theory of “testing in the limit” analogous to a similar concept in inductive inference and learnability theory.

Preliminaries

Our model of testing is based upon the observation that test data is used to distinguish amongst programs (in particular to distinguish between an erroneous program and a correct program). We further observed that testing is a finite process so that only a finite amount of test data can be generated and used in this distinguishability effort. This leads to the following theorem:

*Also affiliated with the Institute for Computer Sciences and Technology of the National Bureau of Standards

Theorem 1. Given any general purpose programming language computing over a countable domain there is no algorithm that given an arbitrary program in the programming language will produce test cases that distinguish the program from all incorrect versions of the program.

Proof: Without loss of generality assume that the countable domain includes a copy of the natural numbers. Consider a program that produces 1 on input 1 and 0 on all other inputs and assume that it is distinguishable from all other programs using some finite test data. We construct a program that computes differently from the above program but which computes the same on the test data. To do this just construct the program that produces 1 on input 1 and 0 on the other test values. On the smallest input not in the test set have the program produce a 1. Both programs operate the same on the test data yet are different.

The proof above involves the observation that the program computes a function and that there are an infinite number of inputs for the program. In particular, there are no references to any undecidable problems; the impossibility of general testing arises solely from the inherent finiteness of testing and the strength of our informal definition of testing.

With the above discouraging theorem, there are a number of ways to proceed. One way is to not be so ambitious and to aim to test programs against specified classes of other programs. This is in fact what occurs in program testing. The tester can only assert that the program is correct with respect to a class of programs if it passes its tests. The class may be functionally specified (9) or specified by some syntactic means (such as programs that can be distinguished by data that ensure all statements are executed [1]). The theory presented below and in [5] takes such an approach.

Another way to proceed is to weaken the intuitive notion of testing to take account of orderings inherent in program classes. Given the ith-program one would search for data that distinguishes that program from preceding programs. This is similar to an approach to inductive inference, inference in the limit, developed by [8]. Since only finite differentiation from a finite set of programs is requested for any particular program, all recursively enumerable sets
of programs that compute total functions are testable (provided functional equivalence of such programs can be determined). We will discuss this latter approach in the final section, but for our initial definition we will follow [5]

**Definitions**

We intend to extract the essentials of testing so we will work with neither representations of functions (programs) nor with functions themselves. We instead present a set based approach to testing that subsumes both (and of course ignores many interesting details). We assume that there is a countable underlying domain D from which a countable class of subsets $S = \{S_i\}$ is constructed, $S_i \subseteq D$. We wish to find a function that when presented a set $S, \in S$ has as value a finite subset, say $R, \in S_i$ with the property that this set distinguishes $S_i$ from every other set in $S$. Distinguishing $S_i$ from every other set in $S$ using $R$ means that the difference, $R - S_i$, is non-empty for every $j \neq i$. Taking the set $S$ to be a collection of total functions, each function represented by its graph, this condition is equivalent to defining a finite function from $S_i$ that differs with every other function of the class for some input value. If there is a function $T$ that given any $S_i$ produces such an $R$ for the class $S$, then $T$ is called a testing function and $S$ is called a testable set.

**Definition 1:** $S$ is said to be testable provided that there exists a function $T$ mapping sets in $S$ to finite subsets of $D$ with the property that for any set $S, \in S$ the difference of $T(S)$ and $R$ for all $R, \in S$ (R $\neq S$) is non-empty (i.e. $(T(S) - R) \neq \emptyset$). Additionally, $T(S) \subseteq S_i$.

As noted above, this definition arises from one given by Cherniavsky and Smith [5] in a study of the relative difficulty of inference versus testing in a recursion theoretic context. Not also that the above definition leads to condition that the symmetric difference of $T(S)$ and $R, R \neq S$ and both $R$ and $S$ in $S$, is non-empty. In the Cherniavsky and Smith definition, testing functions was addressed. By identifying functions with their graphs the above set-based definition captures the same essentials as the earlier functional definition. Finally note that the definition is captures the unique identification of sets by a correct conjunction of positive membership queries. If a putative set $S$ passes all of $T(S)$'s queries, then the set is $S$ or is outside of $S$. Some examples of testable sets follow.

**Example 1:** The pattern languages of Angluin [2] are inferable.

A pattern language is a set of strings defined by all possible substitutions of strings for variables in a pattern consisting of symbols from the string alphabet $\Sigma$ and variables drawn from some countable variable set. Testability follows from the fact that this class of sets is recursively enumerable, each set is recursive, and each set is uniquely determined by a finite subset of patterns (consisting of null substitutions, substitutions of single characters from $\Sigma$, and a simple combination substitution). For more information see [4].

**Example 2:** The class of all finite sets $F$ over a domain $D$ is not testable.

To see this assume that $T$ is a testing function for $F$. Let $A$ be any set in $F$ and let $A_1$ be $T(A)$. $A_1$ is finite and hence in $F$. $A_1$ is included within all finite extensions of $A_1$. Let $B$ be any such extension (there are infinitely many). The difference $A_1 - B$ is empty contradicting the assumption that $T$ is a testing function for $F$. This example is essentially theorem 1.

**Game Formulations of Testing**

The notion a testable class of sets leads naturally to a formulation of testability in terms of games. This formulation gives a different way of talking and thinking about the testing process by formulating it as an adversarial game.

The game for testing is a two person game. The players are simply called Player 1 and Player 2. Player 1 initially chooses a set from a class of sets $S$. Player 1 and Player 2 then alternate plays. Player 1 playing elements from the chosen set and Player 2 playing sets from $S$. Player 1 plays fairly in that each element in the chosen set is eventually played and Player 2 plays fairly in that a set that contains all elements already played by Player 1 must be played by Player 2.

Player 1 wins the testing game provides he forces Player 2 to pick the chosen set. Player 1 has a winning strategy for $S$ provided that for all sets in $S$ Player 1 can win the testing game. Our characterization theorem is that $S$ is testable if and only if Player 1 has a winning strategy in the testing game.

**Theorem 2:** Player 1 has a winning strategy in the testing game for class $S$ of sets if and only if $S$ is testable.

**Proof:** Assume that $S$ is testable with testing agent $T$. For any set $S_i$, Player 1 simply plays elements from $T(S_i)$. Because the differences $T(S_i)$ and $S_i, j \neq i$, are non-empty, by consistency Player 2's only choice is $S_i$ after Player 1 has played $T(S_i)$.

Assume that Player 1 has a winning strategy in the testing game for the class $S$. Define $T(S_i)$ to be the set $A$ played by Player 1 in winning the testing game with chosen set $S_i$. $S$ is testable since if the difference between $T(S_i)$ and $S_i$ for some $i \neq j$ is empty, then Player 2 could have played $S_i$ in the testing game. This contradicts the choice of $A$ that yields a winning game with initial set $S_i$. 

The above theorem, as is clear from its simple proof, follows directly from the definition of testing. However, by reformulating testing in the form of a game we are able to both sharpen our thinking about testing and conveniently pose some complexity questions. We give some examples of sets that are testable or not according to the game theoretic definition.
Example 1: The class of pattern functions is testable. For any particular function, Player 1 plays the tuples consisting of the empty string inputs and resultant pattern, the single symbol string inputs and the resulting patterns, and the two symbol substitutions and resultant patterns. This information uniquely identifies the pattern and the pattern function.

Example 2: The class of pattern sets is not testable. If Player 1 picks the set generated by a single variable then $\Sigma' \cup \{a\}$ is generated. No matter what examples Player 1 plays, Player 2 can always play a pattern set generated by a sufficiently long string of variables.

In the following we restrict our attention to a particularly simple class of sets – finite sets which are subsets of a finite domain. Note that the testing problem has a trivial solution when the class of sets is restricted to range over such finite sets. We consider finite sets that represent functions with a finite domain $\{1, \ldots, n\}$ and a two element range $\{0, 1\}$. We view the functions as characteristic functions on $\{1, \ldots, n\}$ and identify the set represented by the characteristic function with the function. Thus our domain $D$ consists of a set of $n$ elements which we identify with the first $n$ integers. The class $S$ is a subset of the power set $\mathcal{P}(D)$. The complexity of testing a set $A \in S$ is the number of elements that Player 1 must play to force Player 2 to choose $A$.

Theorem 3. The problem of determining whether there exists a $k$-play or less test for set $A$ from $S$ is NP-complete in the size of $A$, $S$, and $k$.

Proof: The proof proceeds by reduction to the hitting set problem shown NP-complete in [6]. The hitting set problem is to determine, given a class $S$ of sets, whether there is a set $S_1$ with $|S_1| \leq k$ such that $S_1$ contains at least one element from each set in $S$. The domain $D$ consists of all integers from 1 to $n$. Consider the complements (with respect to $D$) of all sets in $S$ – call this class of sets $S^c$. Add to $S$ the set $D$ and consider the problem of testing $D$. We claim that $D$ is testable in $k$-moves or less if and only if there is a choice of a set $H$ of $k$ elements or less that is a hitting set for $S^c$. Given the hitting set $H$, $H$ provides sufficient information to test $D$. Each element of $H$ is in $D$ and $H \subseteq S \notin S$ is non-empty for all $S \notin S$ not equal to $D$.

We show that if $D$ is distinguished with a $k$-move game (the moves comprising a set $H$), then $H$ is a hitting set for the complements. Assume that $H$ is not a hitting set and note that if some complement of a set $\overline{S} \in S$ doesn’t have an element in common with $H$, then $H \subseteq S$. But then Player 2 could play either $D$ or $S$ after Player 1 plays $H$. Thus Player 1 did not have a winning strategy playing $H$ contradicting our assumption.

We define the minimum size of a test for a class of sets $S$ to be the sum of the minimal games for each set in $S$. We can represent functions by their graphs – including the functions computed by programs. One easy test for the gross inadequacy of a testing methodology (over a finite class of programs) is to note that the total number of test cases generated is less than the minimum test size. If the test set for an individual program is less than the minimal game for that program then the program is inadequately tested. Thus knowing the minimum test size gives information on the minimal amount of work necessary to distinguish amongst sets of programs.

The above theorem shows that minimality is a difficult property to establish. We are, however, able to obtain some bounds on the length of test games for sets representing finite functions. We are able to show that amongst classes of functions of cardinality $p$ there must be a test game of length $\log_2(p)$ or less; if the minimal length game is at least $k$ for every function in a class of functions, then the cardinality of that class is at least $2^k$; and that there are classes of functions of cardinality $p$ containing a function whose test game is of length $p - 1$. We have been unable to obtain any non-trivial average case results and will close this section with a strong conjecture regarding average test game lengths for classes of functions. We begin with an easy lemma.

Lemma 4. A class of finite functions of cardinality $p$ contains a function with a test game of length at most $\lceil \log_2(p) \rceil$.

Proof: The proof is by divide and conquer. Let $S_0$ be the class of functions. Find some element $a_0$ from the domain of the finite functions such that two functions $f_1$ and $f_2$ differ in value on $a_0$, say $f_1(a_0) = 1$ and $f_2(a_0) = 0$. Let $S_1$ be the smaller of $\{ f \in S | f(a_0) = 0 \}$ and $\{ f \in S | f(a_0) = 1 \}$. Note that the cardinality of $S_1$ is at most half the cardinality of $S_0$. Continue the argument until the resulting set $S_i$ is a singleton. Let $g$ be the sole member of $S_i$. The test game of length $\lceil \log_2(p) \rceil$ consists of the pairs $\{(a, g(a))\}$.

The proof of the theorem relating minimal test games to the cardinality of sets of functions is trickier. Call a class of sets $\Sigma$ $k$-slippery provided that the cardinality of $\Sigma$ is at least two and for all $X \subseteq \Sigma$ and all finite sets $K (|K| = k)$ there is a $Y \subseteq X$ such that $X \setminus K = Y \setminus K$.

Lemma 5. A $k$-slippery class of sets has no test game of length less than $k + 1$.

Proof: The proof follows directly from the definition of $k$-slippery. Any strategy for Player 1 for a length $k$ game requires choosing a $k$ element finite set. By the definition of $k$-slippery, Player 2 can always choose a set different from Player 1's initial choice that also includes the same $k$-elements.

Lemma 6. A $k$-slippery class of sets that is not $k + 1$-slippery has a length $k + 1$ game for some set in the class and no shorter length game.

Proof: The proof follows directly from the definition of $k$-slippery and the previous lemma.

Lemma 7. A $k + 1$-slippery class of sets $S$ is also a $k$-slippery class of sets.
and be one of the functions and form the inductive case. 

\[ f(b) = g(b) \neq f_i(b), \text{ then there must be a fourth function } g_i \text{ that agrees with } f_i \text{ at } b \]  

To see that \( S_1 \) is \( n \)-slippery, let \( X \) be a set in \( S_1 \) and let \( K \) be an arbitrary finite set of size \( n \). We need to find a \( Y \in S_1 \)

\[ X \cap K = Y \cap K. \]

subcase 1: \( (a, f(a)) \in K. \) By definition of \( S_1 \), \( Y \) must be chosen so that \( (a, f(a)) \in Y. \) Since \( S \) is \( n+1 \)-slippery, it is \( n \)-slippery. Set \( X \) and \( K \) as above and choose the \( Y \) from \( S_1 \).

Now \( X \cap K = Y \cap K \). Thus \( (a, f(a)) \in Y \) and thus \( Y \in S_1 \).

subcase 2: \( (a, f(a)) \notin K. \) Augment \( K \) by \( (a, f(a)) \) and use the \( n+1 \)-slipperiness to choose a \( Y \) such that \( X \cap K = Y \cap K \). Now, as in the previous case, \( (a, f(a)) \in Y \cap K \). Thus \( (a, f(a)) \in Y \) and thus \( Y \in S_1 \).

To see that \( S_2 \) is \( n \)-slippery, choose \( X \in S_2 \) and let \( K \) be an \( n \) element finite set.

subcase 1: \( (a, f(a)) \in K. \) Since \( S \) is \( n+1 \)-slippery, it is \( n \)-slippery. Find the \( Y \) resulting from the \( n \)-slipperiness choosing \( X \) and \( K \) as above. From the definition of \( S_2 \), \( (a, f(a)) \notin X. \) Since \( X \cap K = Y \cap K \), \( (a, f(a)) \notin Y. \) Thus \( Y \in S_2 \).

subcase 2: \( (a, f(a)) \notin K. \) Augment \( K \) by adding \( (a, f(a)). \) Now use the \( n+1 \)-slipperiness of \( S \) to find a \( Y \) such that \( X \cap K = Y \cap K. \) Since \( (a, f(a)) \notin X \) and \( (a, f(a)) \notin Y \) and by deleting from \( K \) the value \( (a, f(a)) \), we have found a \( Y \in S_2 \) such that \( X \cap K = Y \cap K. \)

It should be noted that the above proofs require careful construction of "dense" finite functions. Most classes of functions will normally have much shorter games and hence smaller test sets.

The question of lower bounds on the size of test sets is answered above. For some classes of functions, the size of all the test sets is bounded by a constant. The question of upper bounds for test set size is also easily answered. If the functions map \([1...n] \to [0,1]\), then the length of the longest game is no more than \( n \). We prove below that there is a sparse collection of functions \( S \) of cardinality \( n+1 \) in which one function has a minimum testing game of length \( n \).

LEMMA 9. There is a collection \( S \) of functions, where \( |S| = n+1 \), containing a function \( f \) whose shortest testing game is length \( n \).

Proof: For \( 1 \leq i \leq n \), define

\[ f_i(z) = \begin{cases} 
0, & \text{if } z = i; \\
1, & \text{if } z \neq i.
\end{cases} \]

Define \( f_0 \) to be the constant function with value 1. Let \( S = \bigcup_{i=0...m} f_i \).

The testing game for \( f_0 \) requires \( n \) moves because at most one function is eliminated from Player 2's choices for every play made by Player 1. Note that the games for all other \( f_i \)'s are only of length 1.
We close this section with a conjecture concerning the average length of testing games for classes of sets. Let $S$ be a class of sets of cardinality $n$. For each set $S \in S$ let $h_n$ be the length of the minimal game for $S$, with respect to $S$. The average game length for $S$, $\text{AVG}(S)$ is

$$\frac{\sum_{h=1}^{n} h}{n}$$

We have the following conjecture.

Conjecture For all classes of finite sets $S$, $\text{AVG}(S) \leq \log_2 n$.

Testing in the Limit

We now turn to definitional considerations regarding the notions of a testing function and a testable set of functions. The definition used in most of this paper is natural in many circumstances but not in all. To illustrate consider the class of polynomial functions (or programs that compute all and only the polynomial functions). For any fixed degree $k$, $k+1$ points suffice to distinguish all of the $k$ degree polynomials. Thus the $k$ degree polynomials are testable (as is, in fact, the class of at most $k$-degree polynomials). However, the class of all polynomials is not testable (a simple consequence of Theorem 1). This seems unnatural and the definitions below attempt to rectify the situation. These definitions are modeled after those of Gold who noticed a similar problem in the definitions of sets inferable and the inference function.

In his classic paper [8], Gold studied the process of inductive inference in a language learning environment. A major contribution of this paper was the refinement of the notion of learning to a notion of learning in the limit and in the characterization of classes of languages and functions that were learnable in the limit. Briefly, learning in the limit (within the linguistic framework) is the description of an identification process which when presented with a grammatical rule. The process is a limit process in that many inferences are made. Examples of these more restrictive definitions are finite identification in which the learner, after seeing a number of examples, stops requiring examples and generates the grammar (see, for example [7]). An even more restrictive definition is fixed time identification. The learner is presented a fixed number of examples (the learner has no option of requesting more examples) and generates the grammar after seeing those examples.

In a previous paper [4], program testing was investigated using an informal notion that testing and inductive inference were complementary processes. The results obtained were that using a strong notion of testing, there were classes of languages that were inferable in the limit, but not testable. It was also shown that all testable sets were inferable, thus it was shown that testing was difficult (that is that there were fewer testable classes of sets than inferable classes of sets).

Upon close examination, the definition of testing that was used in [8] was more akin to inference by finite identification rather than inference in the limit. When the testing definition is weakened by using a limiting process as an identification criterion, the classes of testable sets become very similar to the classes of inferable sets. In this chapter we make precise the definitions informally given above and we describe, giving examples, testing in the limit, finite testing, and fixed testing.

As above we assume that $D$ is a countable domain and that $S$ is a countable class of sets drawn from the domain $D$. We further assume that the class of sets is enumerable (not necessarily effectively) and the each set in $S$ is enumerable (again not necessarily effectively).

Definition 2: $S$ is inferable in the limit provided there is a function $IM$ such that for any set $S \in S$ there is a finite set $A \subseteq S$ such that $IM(A) = S$ and for all finite $B \supset A$ and $B \subseteq S$, $IM(B) = S$. We call $A$ a sample necessary to infer $S$. The inference function is said to be consistent [10] provided that for all finite subsets $A$, of sets $S \in S$, $A \subseteq IM(A)$. The inference function is said to be normalized provided that $IM$ is consistent and the only sets mapped to an inferred set are samples and the finite sets between samples and the inferred set.

We assume that all of our inferences are consistent inferences. Only minor changes are required otherwise. Similarly we assume that all inference functions are normalized.

This assumption loses no generality since any unnormalized inference function $IM$ can be normalized to an inference function $IM^*$ by defining $IM^*$ on samples and their finite supersets to be the same as $IM$ and $IM^*$ to be the identity function on all other finite sets. Subsequently, references to inference are to be read as references to consistent, normalized inference.

Definition 3 $S$ is finitely inferable provided that $S$ is inferable and the inference function $IM$ is constrained to be either the identity on a finite set or to be the inferred set on a finite set.
Definition 4 $S$ is fixed time inferable if it is inferable and there is a constant $k$ such that on sets with cardinality greater than or equal $k$ the inference function's value is the inferred set. These notions of inferable are akin to the notion of ineffective identification discussed in [6].

We again characterize testable classes of sets by distinguishability criteria. The constraints under which distinguishability information is to be obtained leads to notions of fixed time testable, finite testable, and testable in the limit. Briefly, fixed time testable sets are sets that are distinguishable using an amount of information independent of the set to be distinguished. Finite time testable sets are sets that are distinguishable using an amount of information that may depend upon the set to be distinguished but which is independent of any ordering of the sets. Testable in the limit requires information that may be dependent upon both the set to be distinguished and some underlying ordering of the sets.

We make the same assumptions concerning $D$ and $S$ as above except that in the case of general testability we assume that $S$ can be well ordered. This requires a weak form of the Axiom of Choice. As before we define a function $T$ that maps sets $S \subseteq S$ to finite subsets. The critical property of $T$ is that the finite subset $T(S)$ be sufficient to distinguish $S$ from other sets in the class $S$.

Definition 5: $S$ is testable in the limit provided that for some well ordering of $S$ there exists a function $T$ that maps sets $S, \subseteq S$ to finite subsets with the property that $(T(S_i) - S_j) \neq \emptyset$ for all $j < i$.

Ordinarily the ordering is natural, the sets have nice representations, and the function $T$ is computable. For example, the class of sets can be recursive sets over the integers, the ordering given by a recursively enumerable indexing, and the testing function given by $T$ which maps an index $x$ to the set of minimal values that distinguish the $i^{th}$ set from the previous $j < i$ non-equivalent sets. If the equivalence problem is decidable for the representations of the sets, then the above procedure results in an algorithm since the testing function $T$ is computable.

More interesting situations occur when the representation is both natural and restricted (polynomial functions represented by an equation, finite state transducers represented by their transition diagrams) and there is some natural size metric on the members of the class of sets to be tested based upon their representation. For the polynomials this would be the degree of the polynomial; for finite state transducers this would be the number of states of the minimal equivalent finite state transducer. For testing such classes of sets the "uniformity" of the function $T$ becomes an issue. That is, as the natural metric is increased, does the function $T$ generate distinguishing test sets in the "same fashion" and based on properties of the sets and not their representations. The example of recursively enumerable classes of recursive functions is an example of a “non-uniform” $T$. $T$ is defined solely in terms of the enumeration of the sets (in an exhaustive fashion) and not on any properties of the sets. Examples of "uniform" $T$'s include those defined on regular sets using the size of the minimal accepting automaton as the metric (the test set is then characterized by strings that are twice the number of states in the automaton accepting the set) and those defined on polynomials using the polynomial degree as the metric (the test set just being a set of points and function values that is one more than the degree of the polynomial).

Questions about the computational complexity of $T$ can also be raised based upon the natural metric. Just such questions were posed in [4].

The notion of testable corresponding to fixed time inference is just the definition of testable given by Definition 1. The notion of testable corresponding to fixed time inference is given below.

Definition 6: $S$ is fixed size testable provided that $S$ is testable and that the test function $T$ produces test sets of size $k$ or less for some fixed constant $k$.

Clearly all collections of finite sets with elements drawn from a finite domain are fixed size testable. The class of pattern functions (for patterns with a fixed number of variables over a fixed alphabet) are also fixed size testable.

Conclusions

We have investigated some abstractions of testing. These provide a framework for understanding testing issues in a broad context. In general we do not have a "practical" model of testing. We claim that our work generalises all notions of testing in that it deals solely with the distinguishability that all testing must accomplish. For any of this work to be of use to the working programmer, it must be specific to the programmer's environment. We believe that the best direct applications will be those dealing with finite state models. The notions of testing that we have investigated are particularly suited to such models. We also believe that there are many interesting issues of computational complexity involving testing – we have investigated some of the simplest. Finally, we have not dealt with a wide variety of problems relating to concurrent executions. Our model still fits (albeit uncomfortably) assuming that input/output relations contain some intermediate behaviors. Similar comments apply to programs which are intended to never produce final results (e.g., operating systems).

References

1. W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky


