Probabilistic testing of Boolean functions

or, How to test locally for a global property

by

Krzysztof Michał Majewski

B.Sc., McGill University, 1998

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 6, 2003

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date July 7th 2003

Abstract

Probabilistic testing is a general problem that has fundamental scientific value as well as direct applications to important concepts in theoretical computer science, such as probabilistically checkable proofs (PCP). We consider specifically the testing of Boolean functions, discussing some key existing results and providing new results for several classes of functions.

In particular, we give efficient tests for a Boolean function to be symmetric (invariant under permutation of its variables) and quasi-symmetric (a symmetric function of the variables it actually depends on).

Contents

Chapter 1

Introduction

1.1 Motivation

The problem of testing comes up in the study of self-testing and self-correcting programs [12]. This in turn has applications to probabilistically checkable proofs [1, 2]. Probabilistically checkable proofs are especially important because they have been shown to yield an alternative characterization of the complexity class NP, and this in turn has led to proofs of the intractability of approximation problems [3, 4].

Checking a function (program) P for correctness can be done in two phases. The first phase tests whether P has a given property. If so, the second phase then checks if P is indeed the correct function, using a test that depends on the property holding.

Testing whether a function is a low-degree polynomial [12] and whether various representations of a graph satisfy certain properties [6, 9, 10] are well-studied problems. This work considers only Boolean functions, that is, functions with domain $\{0,1\}^n$ and range $\{0,1\}$. We look at tests for membership in the following families:

- Monotone functions
- Self-dual functions
- Symmetric functions
- Quasi-monadic functions
- Quasi-symmetric functions

We present some previous results from the literature and some new results. In particular, we give efficient tests for a Boolean function to be symmetric (invariant under permutation of its variables) and quasi-symmetric (a symmetric function of the variables it actually depends on).

The underlying theme of this work is how to test locally for a global property.

1.2 What is testing?

1.2.1 Basic definitions

Boolean function

A *Boolean function* is a function $f : \{0,1\}^n \rightarrow \{0,1\}$. We can write Boolean functions as arithmetic expressions over $\mathcal{Z}_2$. For example: $excl(x,y) = x + y$ is the function which takes on value 1 if and only if the inputs x and y differ.

Family

A *family* is a possibly infinite set of functions which possess a given property. One example is the family of monotone functions.

Distance

The *distance* $d(f,g)$ between two functions f and g with domain D is $|\{x \in D.f(x) \neq g(x)\}|$. The distance between function g and family $\mathcal{F}$ is $min_{f \in \mathcal{F}} d(f,g)$. Function f is said to be ϵ-far from (respectively, ϵ-close to) $\mathcal{F}$ if $\frac{d(f,\mathcal{F})}{|D|} \geq$ (respectively, $\leq$) ϵ.

Boolean lattice

Consider the domain $D \equiv \{0,1\}^n$ of binary strings $x_0 x_1 \ldots x_n$. Define the relation $\leq$ on D by $x \leq y \iff \forall i.\{x_i \leq y_i\}$. This is a partial order P on D. The smallest element is the string $0 \ldots 0$, the largest element is the string $1 \ldots 1$. The *Boolean lattice* is the lattice formed by P.

Boolean algebra

A *Boolean algebra* is a Boolean lattice L in which every point $x \in L$ has a *complement* $\overline{x}$, and any two points x, y in the lattice can be combined using a disjunction operator $\vee$ or a conjunction operator $\wedge$ to obtain a new point in L.

Weight

The *weight* of a point in the Boolean lattice is the number of ones it contains.

Row

A *row* of the Boolean lattice is the set containing all points of a given weight, and only those points. The w^{th} row is the row containing exactly the points of weight w.

Sensitivity

Consider the Boolean algebra $D \equiv \{0,1\}^n$ and Boolean function $f : D \to \{0,1\}$. Define $N \equiv \{1, \ldots, n\}$. Let $flip : D \times N \to D$ be the operation which, given $x \in D$ and $i \in N$, flips the i^{th} bit in x to obtain a new point $y \in D$. We say that f is *sensitive* to variable $i \in N$ if there exists a point $x \in D$ such that $f(x) \neq f(flip(x, i))$.

1.2.2 Characterizing functions

We want to know what properties hold of a function. Recall that we seek a local test for a global property, that is, membership in some family $\mathcal{F}$. A *characterization* is a formalization of a property. (A property may have more than one characterization.) We are interested in characterizations to the extent that they yield tests. The following definitions are adapted from [12].

Exact local characterizations

Consider Boolean functions on Boolean algebra D, and k-tuples of input points $(x_1, x_2, \ldots, x_k) \in D^k$. A *$k$-local property* $\mathcal{P}$ is a formula

$$\lambda f.\lambda(x_1, \ldots, x_k).R(x_1, \ldots, x_k, f(x_1), \ldots f(x_k))$$

where $R \subseteq D^k \times \{0,1\}^k$ is some relation on k-tuples of input points and their images under f. For example, the following is a 2-local property:

$$\lambda f.\lambda(x_1, x_2).\{x_1 \leq x_2 \longrightarrow f(x_1) \leq f(x_2)\}$$

A function f *satisfies* k-local property $\mathcal{P}$ when $\mathcal{P}(f)(x_1, \ldots, x_k)$ holds for all $(x_1, \ldots, x_k) \in D^k$. A k-local property $\mathcal{P}$ is an *exact k-local characterization* of a family $\mathcal{F}$ when f satisfies $\mathcal{P}$ iff $f \in \mathcal{F}$. The 2-local property in the above example is an exact local characterization of the family of monotone functions: it is satisfied for all pairs of input points (x_1, x_2) iff f is monotone.

Robust characterizations

Given a k-local characterization $\mathcal{P}$ of $\mathcal{F}$, suppose we sample the k-tuples using a probability distribution ρ on D^k. $\mathcal{P}$ is (ρ, δ, ϵ)-*robust* if, under the distribution ρ, f fails to satisfy $\mathcal{P}$ on at least a weighted fraction δ of the k-tuples when f is ϵ-far from $\mathcal{F}$. In the monotonicity example above, suppose ρ assigns equal probability to those pairs $\{(x, y) : x \leq y\}$ where x and y differ in exactly one bit, and zero probability to all other pairs. It turns out that this yields a $(\rho, \epsilon/n^3, \epsilon)$-robust characterization of the monotone functions (see Section 2.1).

Robustness implies that, on average, the local property $\mathcal{P}$ will have something to say about the global feature, membership in $\mathcal{F}$. In particular, it quantifies the relationship between the local test and the global feature in terms of the relationship between δ and ϵ.

Efficiency

We are interested in characterizations which are not only local, exact, and robust, but also efficient. The complexity of a (ρ, δ, ϵ)-robust characterization is

given by $1/\delta$. This is the number of k-tuples a test must sample for the expected number of $\mathcal{P}$-violating tuples to be 1 if f is indeed ϵ-far from $\mathcal{F}$. In particular, we are interested in characterizations (tests) with complexity polynomial in n, for any n-adic Boolean function to be tested.

Test

A *test* for membership in $\mathcal{F}$ is an implementation of a robust exact local characterization of $\mathcal{F}$. A test must *accept* a function f with high probability if $f \in \mathcal{F}$. [1] A test must *reject* f with high probability if f is ϵ-far from $\mathcal{F}$. Otherwise, the test may accept or reject f arbitrarily. We define *high probability* to be $2/3$. [2]

Query complexity

The *query complexity* of a test is the number of *samples* the test takes. A *sample* is the evaluation of f at all the points in a single k-tuple.

[1] Some definitions, for example [7, 8], require that functions in $\mathcal{F}$ be always accepted. That is, by those definitions, a test cannot produce false negatives.

[2] In fact, this can be any probability greater than $1/2$. To see this, imagine running the test k times. If it accepts most of the time, then f is probably in $\mathcal{F}$. By making k arbitrarily large, we can achieve arbitrarily high accuracy. If we disallow false negatives, we can replace $2/3$ with any non-zero probability.

Chapter 2

Previous results

2.1 Monotone functions

Goldreich et al. [7, 8] present a monotonicity test of complexity poly(n/ϵ). This test samples the function at arguments of its choice. The authors also analyze testing with random examples, and discuss extensions to other domains and ranges. We summarize their main result here.

2.1.1 Result

The test selects a random $x \in \{0,1\}^n$ and $i \in 1, \ldots, n$. We obtain y from x by flipping the i th bit in x. The test rejects if x, y is a monotonicity-violating tuple. Otherwise the steps above are repeated up to n^3/ϵ times. Let $\delta_M(f)$ denote the fraction of such pairs (x, y) which violate monotonicity for a function f. Let $\epsilon(f)$ be the distance of f from the family of monotone functions. The main result is

$$\delta_M(f) \geq \frac{\epsilon(f)}{n^3}$$

from which the correctness of the test follows.

Strategy

The result is proved with the aid of two lemmas. The first lemma shows the existence of a particular matching between two large sets R and S of vertices (points) in the Boolean n-lattice. The weights of the points in R are disjoint from the weights of the points in S. The matching maps vertices x in R to vertices y in S, such that $x \prec y$ but $f(x) > f(y)$. The second lemma shows that

for any such matching there exist vertex disjoint paths from one set to the other. If x and y are the endpoints of such a path, then since $x \prec y$ and $f(x) > f(y)$, it follows that the path contains an edge (x', y') with $x' \prec y'$ and $f(x') > f(y')$. In other words, at some point along the path the value of f decreases while the input increases. Then (x', y') is a monotonicity-violating pair.

The size of the sets R and S is shown to be at at least $(\epsilon_M(f)/2n^2) \cdot 2^n$. Since vertex disjointness implies edge disjointness, and every path contains a violating pair, there are at least $(\epsilon_M(f)/2n^2) \cdot 2^n$ violating edges (pairs). The total number of edges in the Boolean n-lattice is $\frac{1}{2} \cdot 2^n \cdot n$. The fraction of violating edges is then $(\frac{\epsilon_M(f)}{2n^2} \cdot 2^n)/(\frac{1}{2} \cdot 2^n \cdot n) = \epsilon_M(f)/n^3$, which is the result.

Extensions

The authors prove two theorems about monotonicity testing with random examples. The first gives a lower bound on the number of such examples needed by a monotonicity tester. The second shows that this bound is tight up to a poly(n) factor. Recall that when we sample the function f at arguments of our choosing, the number of samples required is $O(\frac{poly(n)}{\epsilon_M(f)})$. In the case of uniformly and independently chosen random examples, however, this number is $\Omega(\sqrt{2^n/\epsilon_M(f)})$.

Also presented are two generalizations of the main result. The first extends the monotonicity test to *unate* functions. A unate function can be viewed as a monotone function with a constant exclusive-or bitmask applied to its inputs. For a monotone function, the output can't decrease if one of the input bits is flipped up (from a zero to a one). Each input bit to a unate function, on the other hand, is either an "up" bit or a "down" bit. The value of the function can't decrease if an "up" bit is flipped up or a "down" bit is flipped down. The extended test has complexity $O(n^{3.5}/\epsilon_M(f))$.

The second generalization treats extended domains and ranges. The complexity of this algorithm scales quadratically with the domain size and linearly with the range.

The paper also poses the problem of whether there exist monotonicity testing algorithms for which the number of required samples is independent of n.

Chapter 3

Results

3.1 Self-dual functions

A Boolean function f over Boolean algebra D is *self-dual* if $\forall x \in D \ \{f(x) = \overline{f(\overline{x})}\}$. Let $\mathcal{S}$ be the set of self-dual functions over D. A function f is *ϵ-far from self-dual* if $d(f, \mathcal{S}) \geq \epsilon$. If we choose $1/\epsilon$ points at random, we expect one *bad* point, that is, a point x such that $f(x) \neq \overline{f(\overline{x})}$. The number k of points that must be chosen to obtain a probability at least $2/3$ of hitting a bad point is given by

$$1 - (1 - \epsilon)^k \quad \geq \quad 2/3$$

or, equivalently,

$$k \quad \geq \quad \left\lceil \frac{\ln 3}{-\ln(1-\epsilon)} \right\rceil.$$

Since $-\ln(1-\epsilon) \geq \epsilon$, it suffices to take

$$k \quad = \quad \left\lceil \frac{\ln 3}{\epsilon} \right\rceil.$$

This is the complexity of our test. Note that this complexity is independent of n.

3.2 Symmetric functions

A function f is *symmetric* if $weight(x) = weight(y) \longrightarrow f(x) = f(y)$. If f is ϵ-far from symmetric, we would have to change its value on at least $\epsilon 2^n$ points

to get a symmetric function. Choose any minimal set of such points, and call these points *bad*.

Now, uniformly choose a point x in the domain. With probability ϵ, x is bad. Now choose (again uniformly) another point y, such that $weight(x) = weight(y)$. We call (x, y) a *symmetric* pair. We call this pair a *violating pair* if $f(x) \neq f(y)$. Consider that at most half the points of a given weight can be bad (otherwise, our set of bad points is not minimal). The probability that (x, y) is a violating pair is then at least $\epsilon/2$.

By the same argument as in the preceding section, it suffices to choose $\left\lceil \frac{\ln 3}{-\ln(1-\epsilon/2)} \right\rceil \leq \lceil \frac{2\ln 3}{\epsilon} \rceil$ symmetric pairs in this fashion to have probability at least $2/3$ of finding a violating pair. This yields a $O(1/\epsilon)$ symmetry test in the obvious way, since a violating pair is a witness to the asymmetry of f.

3.3 Quasi-monadic functions

A Boolean function f is *quasi-monadic* if it is sensitive to at most one of its input variables [5, 11].

We give an $O(n/\epsilon)$ test for quasi-monadicity. First of all, here is an $O(1/\epsilon)$ test which always accepts if f is the projection function $proj(n, i)$, and rejects with high probability if f is ϵ-far from $proj(n, i)$.

Algorithm 3.3.1: PROJ(i, f)

main
  $Result \leftarrow ACCEPT$
  repeat up to $\frac{\ln 3}{\epsilon}$ **times**
    Uniformly choose a point x in the domain
    Let x_i be the value in point x of the i-th variable
    if $f(x) \neq x_i$
      $Result \leftarrow REJECT$
  until $Result = REJECT$
  return $(Result)$

If f is $proj(n,i)$, rejection will never happen, and the test accepts. If f is ϵ-far from $proj(n,i)$, it means that on an ϵ-fraction of the domain, $f(x) \neq proj(n,i)$ and thus $f(x) \neq x_i$. So if we choose $O(1/\epsilon)$ points, we expect to find at least one such "bad" point.

Similarly, here's an $O(1/\epsilon)$ test which always accepts if f is the negated projection function $\neg proj(n,i)$, and rejects with high probability if f is ϵ-far from $\neg proj(n,i)$.

Algorithm 3.3.2: NOTPROJ(i,f)

main
  $Result \leftarrow ACCEPT$
  repeat up to $\frac{\ln 3}{\epsilon}$ **times**
    Uniformly choose a point x in the domain
    Let x_i be the value in point x of the i-th variable
    if $f(x) = x_i$
      $Result \leftarrow REJECT$
  until $Result = REJECT$
  return $(Result)$

Here's an $O(1/\epsilon)$ test which always accepts if f is the constant function $const1$, and rejects with high probability if f is ϵ-far from $const1$.

Algorithm 3.3.3: CONST1(f)

main
  $Result \leftarrow ACCEPT$
  repeat up to $\frac{\ln 3}{\epsilon}$ **times**
    Uniformly choose a point x in the domain
    if $f(x) \neq 1$
      $Result \leftarrow REJECT$
  until $Result = REJECT$
  return $(Result)$

If f is $const1$, rejection will never happen, and the test accepts. If f is ϵ-far from $const1$, it means that on an ϵ-fraction of the domain, $f(x) \neq 1$. So if we choose $O(1/\epsilon)$ points, we expect to find at least one such "bad" point.

Similarly, here's an $O(1/\epsilon)$ test which always accepts if f is the constant function $const0$, and rejects with high probability if f is ϵ-far from $const0$.

Algorithm 3.3.4: CONST0(f)

main
Result $\leftarrow$ *ACCEPT*
repeat up to $\frac{\ln 3}{\epsilon}$ **times**
{
Uniformly choose a point x in the domain
if $f(x) \neq 0$
{*Result* $\leftarrow$ *REJECT*}
}
until *Result* = *REJECT*
return (*Result*)

Now for the quasi-monadicity test.

Algorithm 3.3.5: QUASIMONADICITY(f)

```
main
 Result ← CONST0(f)
 if Result = ACCEPT
   return (Result)
 Result ← CONST1(f)
 if Result = ACCEPT
   return (Result)
 for each variable i in the domain do
 |  Result ← PROJ(i, f)
 |  if Result = ACCEPT
 |    break
 |  Result ← NOTPROJ(i, f)
 |  if Result = ACCEPT
 |    break
 return (Result)
```

If f is quasi-monadic, it must be a constant function, or a projection function, or a negated projection function. In each of these cases, QUASIMONADICITY accepts.

If f is ϵ-far from quasi-monadic, it must be ϵ-far from every quasi-monadic function. Thus, CONST0, CONST1, PROJ(I), and NOTPROJ(I) will all reject, and the test rejects.

CONST0, CONST1, PROJ(I), and NOTPROJ(I) all have running time $O(1/\epsilon)$, since they sample $1/\epsilon$ points and perform an $O(1)$ computation on each point. The *for* loop in QUASIMONADICITY runs at most n times, so QUASIMONADICITY has complexity $O(n/\epsilon)$.

3.4 Quasi-symmetric functions

We now move on to the *quasi-symmetric* functions, that is, functions which are symmetric if we consider only their significant inputs [5, 11].

Given function f on Boolean algebra $D = Pow(N)$, let f' be the section of f to the Boolean algebra $D' = Pow(N')$, where $N' \subseteq N$ contains exactly those variables in N to which f is sensitive. Then f is quasi-symmetric iff f' is symmetric.

Since quasi-symmetry is more general than symmetry, the symmetry test is now too strict: two points with same weight and different image need not be a counterexample (namely, if the points swap a significant bit with a differently valued insignificant bit). In other words, the symmetry test gives false negatives when applied to a quasi-symmetric function.

Our test proceeds in two phases. The first phase attempts to isolate the significant inputs of f. The second phase performs a symmetry test (as in Section 3.2) on a section of f to those variables which have been discovered in the first phase.

Although Phase I may or may not succeed in finding all the significant variables, all the variables in $\mathcal{I}$ are guaranteed to be significant. Since the symmetry test of Phase II is performed on only those variables, there can be no false negatives - f is rejected only if it is *not* quasi-symmetric.

Algorithm 3.4.1: QUASISYMMETRY(f)

main

$\mathcal{I} \leftarrow$ PHASE_I(f)

PHASE_II($f, \mathcal{I}$)

Algorithm 3.4.2: PHASE_I(f)

procedure PHASE_I(f)

$\mathcal{I} \leftarrow \emptyset$

while true

do
- **repeat up to k times**
  - Uniformly choose an assignment w to $\mathcal{I}$
  - Uniformly choose a pair $(w.x, w.y)$ (w need not be a prefix)
- **until** $f(w.x) \neq f(w.y)$
- **if** $f(w.x) = f(w.y)$ **then break**
- **else**
  - $i \leftarrow$ WALK(f, w, x, y)
  - $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$

return ($\mathcal{I}$)

procedure WALK(f, w, x, y)

Walk from x to y until you encounter two points x', y' such that x' differs from y' on exactly one bit i, and $f(w.x') \neq f(w.y')$.

return (i)

Algorithm 3.4.3: PHASE_II(f)

procedure PHASE_II($f, \mathcal{I}$)
Result $\leftarrow$ *ACCEPT*
repeat up to k **times**
$f' \leftarrow$ RESTRICT($f, \mathcal{I}, z$)
Result $\leftarrow$ SYMMETRY(f')
until *Result* $=$ *REJECT*
return (*Result*)

procedure RESTRICT($f, \mathcal{I}, z$)
$D \leftarrow$ Variables of f
assign z **to** $D - \mathcal{I}$
$f' \leftarrow$ Restriction of f given z
return (f')

procedure SYMMETRY(f)
Perform one step of the symmetry test [Section 3.2] on f

It remains to show that the test is complete, in the sense that f is indeed rejected with probability at least $2/3$ if it is ϵ-far from quasi-symmetric. The following analysis will establish the completeness of the test and determine its efficiency.

Phase I maintains as loop invariant the set $\mathcal{I}$ of significant variables found at each step. Let $F_{\mathcal{I}}$ be the family of n-adic functions that depend on only the set $\mathcal{I}$. Let $g \in F_{\mathcal{I}}$ be a function in $F_{\mathcal{I}}$ that is closest to f.

Let $\beta \stackrel{\text{def}}{=} \epsilon/5$. We now consider two cases – Case I, where $d(f, g) > \beta$; and Case II, where $d(f, g) \leq \beta$. In the following, we choose

$$\begin{aligned} k &\equiv \left\lceil \frac{5 \ln 6}{\epsilon} \right\rceil \\ &\geq \left\lceil \frac{\ln 6}{-\ln (1 - \epsilon/5)} \right\rceil . \end{aligned}$$

Consider a given iteration of the outer *while* loop of PHASE_I. Suppose Case I holds at this moment. The probability of reaching the *break* statement in this iteration is simply the probability that $f(w.x) = f(w.y)$ in each of the k trials:

$$\begin{aligned} Pr(break) &= Pr(f(w.x) = f(w.y))^k \\ &= (1 - Pr(f(w.x) \neq f(w.y)))^k \\ &= (1 - d(f,g))^k \\ &\leq (1-\beta)^k \\ &\leq 1/6 \end{aligned}$$

by the above choice of k.

Thus, the probability of breaking out of the outer loop prematurely is acceptably low. In other words, when we eventually break out of the loop, it is likely that we are in Case II, which we now analyze.

Let QS be the set of n-adic quasisymmetric functions. By the triangle inequality, we know that $d(f, QS) \leq d(f,g) + d(g, QS)$. In Case II, we know that $d(f,g)$ is small. Now, we are interested in the situation where f is at least ϵ-far from QS (since we've already shown the test accepts functions in QS, and we're allowed to be wrong for functions that are ϵ-close to QS). So if $d(f, QS)$ is at least ϵ, and $d(f,g)$ is at most β, then $d(g, QS)$ must be at least $\epsilon - \beta = 4\beta$.

Let S be the set of i-adic symmetric functions, where $i \stackrel{\text{def}}{=} |\mathcal{I}|$.

We have $d(g(\cdot, z), S) \geq d(g, QS)$ [1] (since $g(\cdot, z)$ is the same for all z). So $d(g(\cdot, z), S)$ is also at least 4β.

Now,

$$\begin{aligned} d(f,g) &= 2^{-n} \sum_{x \in B_n} f(x) \neq g(x) \\ &= 2^{i-n} \sum_{z \in B_{n-i}} 2^{-i} \sum_{y \in B_i} f(y,z) \neq g(y,z) \end{aligned}$$

[1] The notation $g(\cdot, z)$ refers to the i-adic function obtained by fixing the values for variables not in $\mathcal{I}$ to the bits of z.

$$= 2^{i-n} \sum_{z \in B_{n-i}} d(f(\cdot, z), g(\cdot, z)).$$

Recall that $d(f, g)$ is at most β. By Markov's inequality, the probability of picking a z such that $d(f(\cdot, z), g(\cdot, z)) \leq 2\beta$ is at least $1/2$.

Combining the last two paragraphs with another use of the triangle inequality, we obtain

$$\begin{aligned} d(g(\cdot, z), S) &\leq d(f(\cdot, z), S) + d(f(\cdot, z), g(\cdot, z)), \text{or} \\ d(f(\cdot, z), S) &\geq d(g(\cdot, z), S) - d(f(\cdot, z), g(\cdot, z)) \\ &\geq (4\beta) - 2\beta \\ &= 2\epsilon/5 \end{aligned}$$

with probability at least $1/2$. Weighing this by the probability $1/2$ from the previous paragraph, we obtain $\epsilon/5$ as the lower bound on the probability of rejecting an f ϵ-far from QS (in Case II).

We have already shown there can be no false negatives. Thus the probability that none of the k iterations in Phase II rejects f is at most

$$(1 - \epsilon/5)^k \leq 1/6.$$

There are two ways a false positive might happen:

1. Case I holds at the end of Phase I (and thus we make no claims about the ability of Phase II to reject f), or,

2. Case II holds at the end of Phase I, and yet despite this Phase II fails to reject f.

We have seen that each of these possibilities has probability at most $1/6$. Thus we reject a function that is ϵ-far from QS with probability at least $2/3$.

It is easy to see that procedure WALK has complexity $O(n)$. Consider the n-hypercube. On any path from x to y, $f(x) \neq f(y)$, there must be a pair of points (x', y'), $f(y') \neq f(x')$, such that x' and y' differ on exactly one bit. Thus it is sufficient to arbitrarily pick any such path, and walk along it until one finds (x', y'). The length of a path is at most n.

The outer while loop of Phase I executes $O(n)$ times, since $|\mathcal{I}|$ is at most n. Each inner while loop executes at most $O(kn)$ times, giving a total complexity of $O(n^2)$ for Phase I.

Phase II has complexity $O(k)$, since it consists of at most k iterations of the symmetry test from Section 3.2.

This gives us a grand total of $O(n^2)$. Given that k is $O(1/\epsilon)$, the complexity of the quasi-symmetry test is $O(n^2/\epsilon)$.

Chapter 4

Conclusions

We have considered tests for several common properties of Boolean functions. These tests, by implementing a characterization of the family of functions satisfying the given property, are able to reveal some combinatorial properties of this family which can be interesting in their own right. Moreover, these tests rely on local characterizations of properties which are, by definition, global over the boolean n-lattice. The complexity of a test is a measure of the relationship between this local characterization and the global one.

The complexities of the tests are in all cases dependent on ϵ, the maximum distance beyond which a family and a putative member are no longer considered "close". This is not surprising, since ϵ is the main parameter governing the classification performed by the test – in the language of Probably Almost Correct (PAC) algorithms, it is the value of "Almost". Moreover, this dependence on ϵ is always linear (that is, $O(1/\epsilon)$ is a factor in the complexity).

The complexities of some tests – namely, the monotonicity test of [7, 8], the quasi-monadicity test, and the quasi-symmetry test – are also dependent on n, the arity of the given function. Others (the symmetry test and the self-duality test) are not. This is perhaps more surprising in the case of symmetry than it is for the more obviously trivial self-duality test. It is tempting to ask whether we can classify families of Boolean functions according to whether or not they are amenable to tests independent of n. We have not answered the question in this work.

We note that the quasi-monadicity and quasi-symmetry tests both provide some additional information about the function under scrutiny. The quasi-

monadicity test , if it accepts, also tells us which of the $2n + 2$ quasi-monadic functions the candidate is probably close to. The quasi-symmetry test tells us which variables the candidate is certain to depend on (although this set is not guaranteed to be complete). This is in contrast to the results of [5, 11], which are stronger, but do not yield tests which provide this information.

Bibliography

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Journal of the ACM*, volume 45, pages 501–555. ACM, 1998.

[3] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd Annual Symposium on Foundations of Computer Science*, pages 2–13, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[4] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *Journal of the ACM*, volume 45, pages 70–122. ACM, 1998.

[5] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing juntas. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, pages 103–112, Vancouver, Canada, 16-19 November 2002. FOCS.

[6] O. Goldreich. Combinatorial property testing (a survey). In *EC-CCTR: Electronic Colloquium on Computational Complexity*, 1997. ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1997/TR97-056/index.html.

[7] O. Goldreich, S. Goldwasser, E. Lehman, and D. Ron. Testing monotonicity. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 426–435, Los Alamitos, CA, 8–11 November 1998. IEEE Computer Society.

[8] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorics*, 20:301–337, 2000.

[9] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

[10] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 406–415, El Paso, Texas, 4–6 May 1997.

[11] M. Parnas, D. Ron, and A. Samorodnitsky. Proclaiming dictators and juntas or testing boolean formulae. *ECCC*, 63, 2001.

[12] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, April 1996.