<table>
<thead>
<tr>
<th>項目</th>
<th>内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>On checkers, Self-Testers, and Self-Debuggers</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Mori, Hiroyoshi; Itoh, Toshiya</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (1994), 871: 138-144</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1994-05</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/84044">http://hdl.handle.net/2433/84044</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

京都大学学術情報リポジトリ
KURENAI
Kyoto University Research Information Repository
On Checkers, Self-Testers, and Self-Debuggers

東京工業大学総合理工学研究科 森啓悦 (Hiroyoshi Mori)
東京工業大学総合理工学研究科 伊東利哉 (Toshiya Itoh)

1 Introduction

1.1 Background and Motivation

When a programmer writes a program $P_f$ that computes a function $f$, one of the main difficulties is to mathematically prove that $P_f$ is correct. Blum and Kannan [BK] first attempted to overcome this problem and introduced a notion of "program checking" as an application of interactive proofs [GMR, BM]. Informally, a function $f$ is said to have a (program) checker $C_f$ if (1) when $P_f$ is correct, $C_f$ making calls to $P_f$ outputs with high probability "correct" on any input $x \in \{0,1\}^*$; and (2) when $P_f$ is incorrect, $C_f$ making calls to $P_f$ outputs with high probability "incorrect" on any input $x \in \{0,1\}^*$ such that $P_f(x) \neq f(x)$. Here we note that the output of a checker $C_f$ only verifies the correctness of the output of any program $P_f$ on specified input $x \in \{0,1\}^*$ but does not guarantees the correctness of the program $P_f$ being checked. Then as an extension of checkers, Blum, Luby, and Rubinfeld [BLR] introduced a notion of "self-tester/corrector," and "self-tester/corrector pair," which is a powerful tool in a practical setting. Informally, a function $f$ is said to have a self-tester $ST_f$ if (1) when $P_f$ is tolerably faulty, $ST_f$ making calls to $P_f$ outputs "pass" with high probability; and (2) when $P_f$ is too faulty, $ST_f$ making calls to $P_f$ outputs "fail" with high probability, and a function $f$ is said to have a self-corrector $SC_f$ if when $P_f$ is not too faulty, $SC_f$ making calls to $P_f$ outputs with high probability a correct answer for every input to $f$. In addition, Blum, Luby, and Rubinfeld [BLR] introduced a notion of "self-tester/corrector pair," which is a powerful tool in a practical setting.

Of course, we may expect that every function has a checker, a self-tester, and a self-corrector, however, it is not the case. Indeed, there exists a function $f : \{0,1\}^* \rightarrow \{0,1\}$ that does not have a checker $C_f$ under some complexity-theoretic assumption [Y], [BF], [BG]. As a relationship among checkers, self-testers, and self-correctors, Blum, Luby, and Rubinfeld [BLR] showed that if a function $f$ has a checker $C_f$, then $f$ has a self-tester $ST_f$ (see Theorem 2.8). This implies that the task of self-testers is not harder than that of checkers. On the contrary, it is not known whether or not every function that has a checker has a self-corrector. Then

**Question 1:** Does every function $f$ that has a checker have a self-corrector?

Intuitively, the task of self-correctors seems to be much harder than that of checkers. In section 3, we show that it is indeed the case assuming the existence of oneway permutations.

Blum, Luby, and Rubinfeld [BLR] also showed that if a function $f$ has a self-tester/corrector pair $(ST_f, SC_f)$, then $f$ has a checker $C_f$. It follows from the result above (see Theorem 2.8) that if a function $f$ has a self-tester/corrector pair $(ST_f, SC_f)$, then $f$ has a self-tester $ST_f$. It should be noted that the resulting self-tester $ST_f$ for $f$ could be very reliable but it is somewhat inflexible for a practical purpose, i.e., $ST_f$ only passes completely correct programs $P_f$ for $f$ but does not pass almost correct programs $P_f$ for $f$. Then
• Question 2: How can we make self-testers for a function \( f \) reliable and flexible?

To investigate this problem, we introduce in section 4 a novel notion of “self-debuggers” for \( f \) and then present in subsection 5.1 a way to design a reliable and flexible self-tester for \( f \) from any self-tester/corrector pair for \( f \).

1.2 Results

In this paper, we first present a negative solution to Question 1, i.e., if one-way permutations exist, then there exists a function \( f : \{0,1\}^* \rightarrow \{0,1\} \) that has a checker but does not have a self-corrector (see Theorem 3.1). To demonstrate a positive solution to Question 2, we introduce in section 4 a novel notion of “self-debuggers” and then show that if a function \( f \) has a self-tester/corrector pair \((\overline{ST}_f, \overline{SC}_f)\), then \( f \) has a self-debugger \( SD_f \) (see Theorem 4.2). As the applications of self-debuggers, we show in subsection 5.1 that for any \( 0 < \epsilon_1' < \epsilon_2' \leq \epsilon' < 1 \), if \( \overline{ST}_f \) is an \((\epsilon_1', \epsilon_2')\)-self-tester for \( f \) and \( \overline{SC}_f \) is an \( \epsilon' \)-self-corrector for \( f \), then for any \( \epsilon_1 < \epsilon_2 \) such that \( \epsilon_1 \leq \epsilon' \), there exists an \((\epsilon_1, \epsilon_2)\)-self-tester \( ST_f \) for \( f \) (see Theorem 5.1) and in subsection 5.2 that for any \( 0 < \epsilon_1' < \epsilon_2' \leq \epsilon' < 1 \), if \( \overline{ST}_f \) is an \((\epsilon_1', \epsilon_2')\)-self-tester for \( f \) and \( \overline{SC}_f \) is an \( \epsilon' \)-self-corrector for \( f \), then for any constant \( \delta > 0 \) such that \( \delta \leq \epsilon' \), there exists an \((\epsilon', \delta)\)-self-debugger \( SD_f \) for \( f \) (see Theorem 5.2). The result of Theorem 3.1 captures our intuition that the task of self-correctors seems to be much harder than that of checkers and the results of Theorems 5.1 and 5.2 provides us a powerful tool in a practical setting.

2 Preliminaries

In this section, we first present definitions and terminologies necessary to the subsequent technical discussions and then overview the results known so far.

2.1 Definitions and Terminologies

Let \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) be a function and let \( P_f \) be a program that purports to compute \( f \). In this paper, any program \( P_f \) is assumed to be static, i.e., \( P_f(x) \) is completely determined by the current input \( x \in \{0,1\}^* \) and does not depend on any input previously asked of the program \( P_f \). We use \( P_f(x) \) to denote the output of a program \( P_f \) on input \( x \in \{0,1\}^* \). A program \( P_f \) for \( f \) is said to be correct if \( P_f(x) = f(x) \) for every \( x \in \{0,1\}^* \) and a program \( P_f \) for \( f \) is said to be incorrect if there exists \( w \in \{0,1\}^* \) such that \( P_f(w) \neq f(w) \).

Definition 2.1 [BLR]: A (probabilistic) program \( M \) is said to be an oracle program if it is allowed to make calls to another program that is specified at its run time. We use \( M^A \) to denote an oracle program \( M \) that makes calls to another program \( A \).

Definition 2.2 [BK]: A probabilistic polynomial time oracle program \( C_f \) is said to be a checker for a function \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) if for every program \( P_f \) that purports to compute \( f \), \( C_f \) on input \( x \in \{0,1\}^* \) satisfies the following conditions:

1. If \( P_f \) is correct, then \( \Pr[C_f^P(x) = \text{"correct"}] \geq 2/3 \) for any \( x \in \{0,1\}^* \);
2. For any \( x \in \{0,1\}^* \) such that \( P_f(x) \neq f(x) \), \( \Pr[C_f^P(x) = \text{"incorrect"}] \geq 2/3 \),

where the probabilities are taken over all possible coin tosses of \( C_f \).
Note that in Definition 2.2, the output of $C_{f}^{P_{f}}$ on any input $x \in \{0,1\}^{*}$ such that $P_{f}(x) = f(x)$ is not specified when $P_{f}$ is incorrect.

We say that a function $f$ is checkable (or $f$ has a checker) if there exists a checker $C_{f}$ for $f$ and also that a language $L$ is checkable (or $L$ has a checker) if there exists a checker $C_{L}$ for its characteristic function $\lambda_{L}$. A function $f$ (or language $L$) is uncheckable if it is not checkable.

Let $N$ be a set of positive integers and let $I \subseteq \{0,1\}^{*}$. Let $I_{1}, I_{2}, \cdots \subseteq I$ be a sequence of subsets of $I$ that satisfies $I_{1} \cup I_{2} \cup \cdots = I$. Note that each $n \in N$ indicates the “size” of each $x \in I_{n}$. We use $D = \{D_{n} \mid n \in N\}$ to denote an ensemble of probability distributions such that $D_{n}$ is a distribution on $I_{n}$ for each $n \in N$. Let $P_{f}$ be a program that purports to compute a function $f$. We use $\text{Err}(P_{f}, f, D_{n})$ to denote the error probability that $P_{f}(x) \neq f(x)$ when $x$ is randomly chosen in $I_{n}$ according to $D_{n}$ for $n \in N$. Let $0 < \beta < 1/2$ be a confidence parameter.

**Definition 2.3** [BLR]: A probabilistic polynomial time oracle program $ST_{f}$ is said to be an $(\epsilon_{1}, \epsilon_{2})$-self-tester for a function $f$ with respect to $D = \{D_{n} \mid n \in N\}$ if for some constants $0 \leq \epsilon_{1} < \epsilon_{2} \leq 1$ and any program $P_{f}$ that purports to compute $f$, $ST_{f}$ on input $1^{n}$, $0 < \beta < 1/2$ satisfies the following conditions:

1. If $\text{Err}(P_{f}, f, D_{n}) \leq \epsilon_{1}$, then $\Pr\{ST_{f}^{P_{f}}((1^{n}, \beta)) = \text{"pass"}\} \geq 1 - \beta$;
2. If $\text{Err}(P_{f}, f, D_{n}) \geq \epsilon_{2}$, then $\Pr\{ST_{f}^{P_{f}}((1^{n}, \beta)) = \text{"fail"}\} \geq 1 - \beta$,

where the probabilities are taken over all possible coin tosses of $ST_{f}$.

Note that in Definition 2.3, the output of $ST_{f}$ is not specified when $\epsilon_{1} < \text{Err}(P_{f}, f, D_{n}) < \epsilon_{2}$. Thus the value $\delta = \epsilon_{2} - \epsilon_{1}$ should be as close as possible to $0$ so that $ST_{f}$ is reliable.

We say that a function $f$ is weakly/strongly self-testable (or $f$ has a weak/strong self-tester) if for some/eny ensemble of distributions $D$, there exist some constants $0 \leq \epsilon_{1} < \epsilon_{2} \leq 1$ such that $f$ has an $(\epsilon_{1}, \epsilon_{2})$-self-tester $ST_{f}$ with respect to $D$ and we also say that a language $L$ is weakly/strongly self-testable (or $L$ has a weak/strong self-tester) if for some/eny ensemble of distributions $D$, there exists some constants $0 \leq \epsilon_{1} < \epsilon_{2} \leq 1$ such that its characteristic function $\lambda_{L}$ has an $(\epsilon_{1}, \epsilon_{2})$-self-tester $ST_{L}$ with respect to $D$. A function $f$ (or a language $L$) is said to be strongly/weakly self-extendable if it is not weakly/strongly self-testable.

**Definition 2.4** [BLR]: A probabilistic polynomial time oracle program $SC_{f}$ is said to be an $\epsilon$-self-corrector for a function $f$ with respect to $D = \{D_{n} \mid n \in N\}$ if for some constant $0 < \epsilon < 1$ and any program $P_{f}$ that purports to compute $f$, $SC_{f}$ on input $1^{n}$, $x \in I_{n}, 0 < \beta < 1/2$ satisfies the following condition: If $\text{Err}(P_{f}, f, D_{n}) \leq \epsilon$, then $\Pr\{SC_{f}^{P_{f}}((1^{n}, x, \beta)) = f(x)\} \geq 1 - \beta$.

We also say that a function $f$ is weakly/strongly correctable (or $f$ has a weak/strong self-corrector), that a language $L$ is weakly/strongly correctable (or $L$ has a weak/strong self-corrector), and that a function $f$ (resp. a language $L$) is strongly/weakly self-uncorrectable in a way similar to the case of self-testers.

**Definition 2.5** [BLR]: A pair of probabilistic polynomial time oracle programs $(ST_{f}, SC_{f})$ is said to be a self-tester/corrector pair for a function $f$ if for some constants $0 \leq \epsilon_{1} < \epsilon_{2} \leq \epsilon < 1$ and an ensemble of distributions $D$, $ST_{f}$ is an $(\epsilon_{1}, \epsilon_{2})$-self-tester for $f$ with respect to $D$ and $SC_{f}$ is an $\epsilon$-self-corrector for $f$ with respect to $D$.

We say that a function $f$ (resp. a language $L$) has a self-tester/corrector pair if there exists a self-tester/corrector pair for $f$ (resp. the characteristic function $\lambda_{L}$ of $L$).

For each $n \geq 0$, let $S_{n} = \{0,1\}^{n}$ and let $R_{n}$ be a set of all possible sequences of coin tosses of a probabilistic polynomial (in $n$) time algorithm on input of length $n$. 

Definition 2.6 (Oneway Permutation): If a function \( g : \{0,1\}^* \mapsto \{0,1\}^* \) satisfies that

1. For any \( x \in \{0,1\}^* \), \( |g(x)| = |x| \) and \( g \) is 1-1 and onto;
2. For any \( x \in \{0,1\}^* \), \( g(x) \) can be evaluated in polynomial (in \(|x|\)) time;
3. For any probabilistic polynomial (in \( n \)) time algorithm \( A \) and each constant \( c > 0 \), there exists a constant \( N_c \geq 0 \) such that \( \Pr\{g(A(y)) = y\} < n^{-c} \) for every \( n > N_c \) and every \( y \in \{0,1\}^n \), where the probabilities are taken over \((y,r) \in S_n \times R_n\),

then we say that the function \( g : \{0,1\}^* \mapsto \{0,1\}^* \) is a oneway permutation.

In the rest of this paper, we sometimes use \( g_n : \{0,1\}^n \mapsto \{0,1\}^n \) for each \( n \in N \) to denote the restriction of a oneway permutation \( g : \{0,1\}^* \mapsto \{0,1\}^* \).

The following inequality is useful in section 5.1 to approximate the error probability (with respect to \( D \)) of a program \( P_f \) that purports to compute \( f \).

Definition 2.7 [S]: Let \( X_1, X_2, \ldots, X_n \) be independent identically distributed 0-1 random variables and let \( \Pr(X_i = 1) = p \) for each \( i \) (\( 1 \leq i \leq n \)). Then

1. \( \Pr\left\{ \frac{1}{n} \sum_{i=1}^{n} X_i - p \leq \delta \right\} \geq 1 - \exp\left\{ -\frac{\delta^2 n}{2} \right\}; \)
2. \( \Pr\left\{ \frac{1}{n} \sum_{i=1}^{n} X_i - p \geq -\delta \right\} \geq 1 - \exp\left\{ -\frac{\delta^2 n}{2} \right\}, \)

for any integer \( n \geq 1 \) and any constant \( \delta > 0 \).

2.2 Known Results

Here we overview the known results on checkers, self-testers, and self-correctors. The following theorems are the relationships among checkers, self-testers, and self-correctors.

Theorem 2.8 [BLR], [R]: If a function \( f : \{0,1\}^* \mapsto \{0,1\}^* \) has a checker \( C_f \), then for any constant \( 0 < \varepsilon_2 \leq 1 \), the function \( f \) has a \((0,\varepsilon_2)\)-self-tester \( ST_f \) with respect to any polynomial time samplable ensemble of distributions \( D \).

Theorem 2.9 [BLR]: If a function \( f : \{0,1\}^* \mapsto \{0,1\}^* \) has a self-tester/corrector pair \( (ST_f, SC_f) \), then the function \( f \) has a checker \( C_f \).

3 Checkable But Self-Uncorrectable Languages

In this section, we give a solution to Question 1 in subsection 1.1, i.e., there exists a weakly self-uncorrectable language \( L \) that is checkable under a complexity-theoretic assumption.

Theorem 3.1: If oneway permutations exist, then there exists a weakly self-uncorrectable language \( L \notin \mathcal{BPP} \) that has a checker.

Proof: Let \( I = \{(0,1)^n \times (0,1)^n \times (0,1)^n \mid n \in N\} \) and let \( I_n = \{(0,1)^n \times (0,1)^n \times (0,1)^n \} \) for each \( n \in N \). We use \( \mathcal{U} = \{U_n \mid n \in N\} \) to denote an ensemble of uniform distributions, i.e., for each \( n \in N \), \( U_n \) is a uniform distribution on \( I_n \), and use \( a \cdot b \) to denote the inner product of \( a, b \in \{0,1\}^n \) modulo 2. Let \( g : \{0,1\}^* \mapsto \{0,1\}^* \) be a oneway permutation. Define a language \( L_g \subseteq I \) to be \( L_g = \{(y,p,0^{|y|}) \in I \mid g^{-1}(y) \cdot p = 1\} \). It is obvious that \( L_g \in \mathcal{NP} \). Note that the definition of \( L_g \) is inspired by the hard-core predicate due to Goldreich and Levin [GL].
We first show that $L_g$ has a checker $C_g$. But there isn’t enough space to show that. So we omit the proof.

Next, we show that $L_g$ is weakly self-uncorrectable. Indeed, we show that for some ensemble of distributions $\mathcal{D} = \{D_n \mid n \in \mathbb{N}\}$ and for any constant $0 < \epsilon < 1$, the characteristic function $\lambda_g$ of $L_g$ does not have an $\epsilon$-self-corrector with respect to $\mathcal{D} = \{D_n \mid n \in \mathbb{N}\}$.

Assume that $L_g$ is strongly self-correctable, i.e., for any ensemble of distributions $\mathcal{D}$ and some constant $0 < \epsilon < 1$, $L_g$ has an $\epsilon$-self-corrector $SC^g_\epsilon$ with respect to $\mathcal{D} = \{U_n \mid n \in \mathbb{N}\}$, where $U_n$ is a uniform distribution on $I_n$ for each $n \in \mathbb{N}$. Define a program $\tilde{P}_g$ that purports to compute $\lambda_g$ in a way that on input $(y, p, z) \in I_n$,

$$\tilde{P}_g((y, p, z)) = \begin{cases} 0 & z \neq 0^n; \\ 1 & z = 0^n. \end{cases}$$

Here we note that $\tilde{P}_g$ errs only for $(y, p, 0^n) \notin L_g \cap I_n$. From the assumption that $g_n$ is a (oneway) permutation, it follows that for each $n \in \mathbb{N}$, $\text{Err}(\tilde{P}_g, \lambda_g, U_n)$ is given by

$$\text{Err}(\tilde{P}_g, \lambda_g, U_n) = \frac{\|\{(y, p, 0^n) \in I_n \mid g^{-1}(y) \cdot p = 0\}\|}{\|I_n\|} = \frac{2^{2n-1} + 2^{n-1}}{2^{3n}} = \frac{2^n + 1}{2^{2n+1}} < \frac{1}{2^n},$$

where $\|A\|$ denotes the cardinality of a finite set $A$.

Then for every $n \geq \lceil \log \epsilon^{-1} \rceil$, $\text{Err}(\tilde{P}_g, \lambda_g, U_n) \leq \epsilon$. This implies that for every $n \geq \lceil \log \epsilon^{-1} \rceil$, $SC^g_\epsilon$ making calls to the program $\tilde{P}_g$ outputs 1 with probability at least $1 - \beta$ if $(y, p, z) \in L_g$ and outputs 0 with probability at least $1 - \beta$ if $(y, p, z) \notin L_g$. Thus we can make this probabilistic polynomial (in $n$) time program $\tilde{P}_g$ such that $\tilde{P}_g((y, p, z)) = \lambda_g((y, p, z))$ with probability at least $1 - \beta$ from $C_g$ and $\tilde{P}_g$. Since $0 < \beta < 1/2$ is a constant (see Definition 2.4), $L_g \in BPP$ [BDG]. By a standard technique (see [BDG]), we assume without loss of generality that there exists a probabilistic polynomial (in $n$) time algorithm $G$ such that $G((y, p, z)) = \lambda_g((y, p, z))$ with probability at least $1 - 2^{-n}$. Then let us consider the following probabilistic algorithm $Inv_g$:

**Inverting Algorithm $Inv_g$:**

**Input:** $y \in \{0,1\}^n$.

I-0: $x := \varepsilon$ (null string); $j := 1$.

I-1: Choose randomly $p \in \{0,1\}^n$.

I-2: Run $G$ on input $(y, p, 0^n)$ to get $b = G((y, p, 0^n))$.

I-3: Run $G$ on input $(y, p \oplus e_j^n, 0^n)$ to get $b_j = G((y, p \oplus e_j^n, 0^n))$.

I-4: If $b = b_j$, then $x := x || 0$ and $j := j + 1$; otherwise $x := x || 1$ and $j := j + 1$.

I-5: If $j \leq n$, then go to step I-3; otherwise continue.

I-6: If $y = g(x)$, then halt and output $x \in \{0,1\}^n$; otherwise halt and output "⊥.”

It is obvious that if $G$ correctly returns $b$ and $b_j$ for each $1 \leq j \leq n$, then $Inv_g$ successfully finds in polynomial time $x \in \{0,1\}^n$ such that $y = g(x)$ (see [GL]). Thus the probability $P_{\text{succ}}$ that on input $y \in \{0,1\}^n$, $Inv_g$ outputs $x \in \{0,1\}^n$ such that $y = g(x)$ is bounded by

$$P_{\text{succ}} \geq (1 - 2^{-n})^{n+1} \geq 1 - (n + 1) \cdot 2^{-n}.$$  

This contradicts the assumption that $g$ is a oneway permutation (see Definition 2.6). Then it follows that $L_g \notin BPP$ and this implies that $L_g$ must be weakly self-uncorrectable.

Thus if oneway permutations exist, then there exists a weakly self-uncorrectable language $L \notin BPP$ that has a different checker. \blacksquare
4 Self-Debuggers

In this section, we present a new notion of "self-debuggers." Informally, a function \( f \) is said to have a self-debugger \( SD_f \) with respect to \( D \) if \( SD_f \) transforms a faulty program \( P_f \) for \( f \) with respect to \( D \) to a less faulty (deterministic) program \( Q_f \) for \( f \) with respect to \( D \).

**Definition 4.1:** A probabilistic polynomial time oracle program \( SD_f \) is said to be a \((\delta, \gamma)\)-self-debugger for a function \( f \) with respect to \( D = \{D_n \mid n \in \mathbb{N}\} \) if for some constants \( 0 \leq \gamma \leq \delta \leq 1 \), \( SD_f \) on input \( 1^n \), \( 0 < \beta < 1/2 \) transforms any program \( P_f \) that purports to compute \( f \) to a (deterministic) program \( Q_f \) for \( f \) and it satisfies the following conditions:

1. If \( \text{Err}(P_f, f, D_n) \leq \delta \), then \( \Pr\{\text{Err}(Q_f, f, D_n) \leq \gamma\} \geq 1 - \beta \);
2. If \( \text{Err}(P_f, f, D_n) > \delta \), then \( \Pr\{SD^{P_f}_f((1^n, \beta)) = \text{"fail"} \cup \text{Err}(Q_f, f, D_n) \leq \gamma\} \geq 1 - \beta \),

where the probabilities are taken over all possible coin tosses of \( SD_f \).

The following theorem shows that if a function \( f \) has a self-tester/corrector pair, then there exists a self-debugger for \( f \).

**Theorem 4.2:** For some constants \( 0 < \epsilon'_1 < \epsilon'_2 \leq \epsilon' < 1 \) and a polynomial time samplable ensemble of distributions \( D \), let \( \langle \overline{ST}_f, \overline{SC}_f \rangle \) is a self-tester/corrector pair for a function \( f \), i.e., \( \overline{ST}_f \) is an \((\epsilon'_1, \epsilon'_2)\)-self-tester for \( f \) with respect to \( D \) and \( \overline{SC}_f \) is an \( \epsilon' \)-self-corrector for \( f \) with respect to \( D \). Then there exists an \((\epsilon', \epsilon''_1)\)-self-debugger \( SD_f \) for \( f \) with respect to \( D \).

5 Applications of Self-Debuggers

In this section, we present two applications of self-debuggers by using Theorem 4.2. The first application is to design a reliable and flexible self-tester from any self-tester/corrector pair and the second application is to design strong self-debugger from any self-tester/corrector pair.

5.1 Making Self-Testers Reliable and Flexible

Assume that a self-tester/corrector pair \( \langle \overline{ST}_f, \overline{SC}_f \rangle \) for a function \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) is given. Then it follows from Theorem 2.9 that \( f \) has a checker \( C_f \) and it follows from Theorem 2.8 that for any constant \( 0 < \epsilon_2 \leq 1 \), \( f \) has a \((0, \epsilon_2)\)-self-tester \( ST_f^{\epsilon_2} \) with respect to any polynomial time samplable ensemble of distributions \( D \). The resulting self-tester \( ST_f^{\epsilon_2} \) can be highly reliable when \( \epsilon_2 \) is taken to be very small. On the other hand, \( ST_f^{\epsilon_2} \) is somewhat of limited use, because it only passes completely correct programs \( P_f \) for \( f \). We often wish to design a flexible self-tester \( ST_f \) for \( f \) in such a way that \( ST_f \) passes almost correct programs \( \tilde{P}_f \) for \( f \). In this section, we give (as a solution to Question 2) a way to design a reliable and flexible self-tester for \( f \) from any self-tester/corrector pair for \( f \).

**Theorem 5.1:** For some constants \( 0 < \epsilon'_1 < \epsilon'_2 \leq \epsilon' < 1 \) and a polynomial time samplable ensemble of distributions \( D \), let \( \langle ST_f, \overline{ST}_f \rangle \) is a self-tester/corrector pair for a function \( f \), i.e., \( ST_f \) is an \((\epsilon'_1, \epsilon'_2)\)-self-tester for \( f \) with respect to \( D \) and \( \overline{ST}_f \) is an \( \epsilon' \)-self-corrector for \( f \) with respect to \( D \). Then for any constants \( \epsilon_1 < \epsilon_2 \) such that \( \epsilon_1 \leq \epsilon' \), there exists an \((\epsilon_1, \epsilon_2)\)-self-tester \( ST_f \) for \( f \) with respect to \( D \).
5.2 Amplification of Self-Debuggers

From Theorems 4.2 and 5.1, we can show the following theorem:

Theorem 5.2:
For some constants $0 < \epsilon_1' < \epsilon_2' \leq \epsilon' < 1$ and a polynomial time samplable ensemble of distributions $D$, let $0 < \epsilon_1' < \epsilon_2' \leq n\epsilon' < 1$ be constants and let $D$ be a polynomial time samplable ensemble of distributions. $(\overline{ST}_f, \overline{ST}_j)$ is a self-tester/corrector pair for a function $f$, i.e., $\overline{ST}_f$ is an $(\epsilon_1', \epsilon_2')$-self-tester for $f$ with respect to $D$ and $\overline{SC}_f$ is an $\epsilon'$-self-corrector for $f$ with respect to $D$. Then for any constant $\delta > 0$ such that $\delta \leq \epsilon'$, there exists an $(\epsilon', \delta)$-self-debugger $SD_f$ for $f$ with respect to $D$.

Proof: This can be led from Theorem 5.1 and Theorem 4.2 easily.

Acknowledgments

The authors wish to thank Ronitt Rubinfeld for her helpful discussion on the relation between checkers and self-testers [R].

References


