Reliable Computing 2 (2) (1996), pp. 139–145

Self-correcting polynomial programs

Guevara Noubir and Henri J. Nussbaumer

In this paper, we introduce a new self-correction algorithm that requires less queries than the simple majority vote. We also introduce new random self-reducibility formulas.

Автокорректирующие полиномиальные программы

Г. Нубир, Г. Нуссбаумер

Предлагается новый алгоритм автокоррекции, требующий меньшего количества проверок, чем простое принятие решения большинством голосов. Вводятся также новые формулы случайной автосводимости.

1. Introduction

The theory of self-testing/correcting programs by random self-reducibility (RSR) [1, 2, 5] is a novel and powerful tool to approach the problem of program correction. It allows real-time testing and correcting of programs. Moreover, it allows the simultaneous checking of the hardware and software without requiring any knowledge of the implementation of a program. Basically, a function is random self-reducible of order k if its value at a given point can be efficiently reconstructed from its evaluation at k random points. The research conducted in this domain focuses on two main issues: the *enlargement* of the set of random self-reducible functions and the *reduction* of the complexity of self-correction schemes based on RSR. In this paper, we are interested in reducing the complexity of the self-correction scheme for multi-variate polynomials. We investigate two methods to achieve the complexity reduction: 1) by reducing the order of RSR in some special cases (e.g., studying polynomials defined over extension fields or introducing some determinism in the choice of the queries) and 2) by substituting a new generic self-correction algorithm to the simple majority vote.

In Section 2, we recall some known results on the RSR of polynomials. In Section 3, we give a lower bound of RSR of polynomials. In Section 4, we introduce new random self-reducibility formulas. In Section 5, we introduce a new algorithm for the self-correction of polynomial programs. The proofs of the theorems can be found in [3].

2. Polynomials random self-reducibility property

The random self-reducibility property allows the reconstruction of the value of a function f by using its evaluation on a finite number of elements taken from its domain of definition. The reconstruction procedure has to be simpler and faster than the direct evaluation of f. In this section, we recall some general results established in [1, 2].

Definition 1. *A function f defined over a set D is said to be random self-reducible of order k if and only if there exists a function φ and a set of probabilistic functions $\sigma_1, \ldots, \sigma_k$ such that:*

$$\forall x \in D,\ \forall r \in D; \quad f(x) = \varphi\Big(r, x, f\big(\sigma_1(r,x)\big), \ldots, f\big(\sigma_k(r,x)\big)\Big). \tag{1}$$

Functions $\sigma_i(\cdot, x)$ have a well defined distribution, and functions φ and σ_i can be computed efficiently.

A large class of mathematical functions have been proven random self-reducible (e.g., fast Fourier transform, matrix permanent, trigonometric functions, etc.). In the rest of this paper, we concentrate on polynomial functions (i.e., functions for computing univariate and multivariate polynomials). Lipton [2] and Blum et al. [1] have studied polynomial functions and have shown that polynomials of degree[1] d are random self-reducible of order $d+1$. This is illustrated in the following theorem:

Theorem 1 [Lipton, Blum et al.]. *For every set $\{a_1, \ldots, a_{d+1}\}$ of pairwise different elements of a finite field F such that $|F| > d+1$, there exists a finite set of elements $\{c_1, \ldots, c_{d+1}\}$ such that for every polynomial P of degree d, defined over F, the following holds:*

$$\forall x, r \in F; \quad P(x) = \sum_{i=1}^{d+1} c_i P(x + a_i r). \tag{2}$$

In this case, the functions σ_i are defined by $\sigma_i(r,x) = x + a_i r$. If $(a_1, \ldots, a_{d+1}) = (1, 2, \ldots, d+1)$, then $c_i = (-1)^{i+1} C_{d+1}^i$, $i = 1, \ldots, d+1$.

Generic program for self-correcting polynomials. The random self-reducibility property of functions can be used to improve the reliability of a program that has an error probability p. Let f be the function computed by a given program such that $\text{Prob}[f(x) \neq P(x)] > p_{\text{err}}$. Blum et al. give an algorithm that reduces the error probability from $\frac{1}{4(d+1)}$ to any value β. We recall this algorithm and we show that it can be generalized to reduce the error probability from $\frac{p}{d+1}$ $(p < 1/2)$ to β. Note that all what needs to be known about P is that it is a polynomial of degree d.

Algorithm 1. *Generic self-correction program(f)*

$N \leftarrow 24 \ln(1/\beta)$
For $m = 1, \ldots, N$ **Do**
  Choose $r \in_U$ F
  $r_m \leftarrow \varphi\Big(r, x, f\big(\sigma_1(r,x)\big), \ldots, f\big(\sigma_k(r,x)\big)\Big)$
Result $\leftarrow$ Majority$\{r_m | \ m : 1, \ldots, N\}$
end

Theorem 2 [Blum et al.]. *If the program f has a probability of error less than or equal to $\frac{1}{4(d+1)}$, then the result given by Algorithm (1) has an error probability less than β.*

Theorem 3. *Given a program f with error probability $\frac{p}{d+1} < \frac{1}{2(d+1)}$, the self-correction algorithm using the vote will have an error probability of β and will make $(d+1) \times \frac{2(1-p)}{(p-1/2)^2} \ln \frac{1}{\beta}$ queries.*

[1] For multivariate polynomials, the degree is the maximum sum of degrees of variables in the same monomial.

3. Lower bound on self-reducibility

Random self-reducibility gives a rigorous method to improve the reliability of programs. Unfortunately, for polynomials of large degree, this method requires too many queries to the defective program since the number of queries is proportional to the degree of the polynomial. To improve the efficiency of this method, two directions can be investigated: 1) the reduction of the order of random self-reducibility of polynomials, and 2) the improvement of the self-correction algorithm. In this section, we give a lower bound to the order of random self-reducibility of polynomials.

To reconstruct the value of a polynomial by random self-reducibility, we need to query a program on $d+1$ points. One may think that, under some conditions, this order (taken equal to $d+1$) can be reduced. These conditions could be that the queries no longer obey a uniform distribution but are deterministically chosen, or that there exists some restrictions on the working field. We show that such a reduction is not possible.

Lemma 1. *Let F be a finite field such that $|F| > d$. No functions $\sigma_1, \ldots, \sigma_d$ exist such that there exists an element $x_0 \in F$ and for every univariate polynomial P of degree d we have: $P(x_0) = \varphi\Big(x_0, P\big(\sigma_1(x_0)\big), \ldots, P\big(\sigma_d(x_0)\big)\Big)$. The only condition on function σ_i is that $\sigma_i(x_0) \neq x_0$.*

Lemma 1 proves that even self-reduction over one point x_0 and of order less than $d+1$ is not possible. This is true regardless of the distribution of the queries and of the working field.

Theorem 4. *The order of self-reduction of univariate polynomial functions of degree d is exactly equal to $d+1$.*

The order of random self-reducibility of multivariate polynomial functions of combined degree d is exactly equal to $d+1$.

The order of self-reducibility of multivariate polynomials of degree d for each variable is equal to $d+1$.

4. Other random self-reducibility formulas

The random self-reducibility property of polynomials follows from Lagrange interpolation. In Section 2, we presented this property as it was first introduced in [1]. In this section, we introduce three novel formulas to reconstruct a polynomial value at point x from $d+1$ uniformly distributed points. These formulas follow from Lagrange interpolation [4].

Consider the set of polynomials of degree d defined over a finite field F. Let w be a root of unity of order $d+1$; w exists if and only if $d+1$ divides[2] $|F|-1$. Lagrange interpolation over the roots of unity gives us a new formula for RSR of reduced computation complexity:

$$\forall x, r \in F; \quad P(x) = \frac{1}{d+1} \sum_{i=0}^{d} P(x + rw^i). \tag{3}$$

[2] If $d+1$ does not divide $|F|-1$, we may consider the polynomials as polynomials of degree d'; where $d'+1$ is the smallest divisor $\geq d+1$ of $|F|-1$.

The other two RSR-formulae are the alternate self-reducibility and the Lagrange interpolation over a sequence of consecutive elements:

$$\forall x, r \in F; \quad P(x) = \frac{-1}{d+1} \sum_{u \neq x + rw^i} P(u), \tag{4}$$

$$\forall x, r \in F; \quad P(x) = \frac{1}{k!} \sum_{i=1}^{d+1} (-1)^{i-1} \frac{i}{k+i} C_{d+1}^{i} P\Big(x + r(k+i)\Big). \tag{5}$$

5. Self-correction with early error detection

To improve the polynomials self-correction scheme, we showed that, according to Theorem 4, it is not possible to reduce the order of random self-reducibility. The only remaining possibility is to find new simple algorithms that are based on random self-reducibility and are more efficient than the ones known in the literature. Sudan and Gemell [6] studied this problem and found new algorithms that tolerate more errors in the defective program. However, these algorithms suffer from an increased complexity. We propose a simple algorithm that acts as a filter. It queries the defective program over a set of points and discards them when it detects some errors. Otherwise, it reconstructs, by RSR, a value from them to be used during the vote phase.

Let f be a program that evaluates a polynomial P of degree d, with error probability $\frac{p}{d+1}$. Consider the set $\{f(x_i) \mid i : 1 \ldots n\}$. When $x_i = w^i$ [5], we are in presence of a Bose-Chaudhuri-Hocquenghem (BCH) code, which can be error-corrected by using a decoding algorithm of BCH codes. When x_i covers all the field F, we have a Reed-Solomon decoding problem [6]. A major disadvantage of these methods is the complexity of the algorithms.

Let $f(x+r), f(x+2r), \ldots, f\Big(x + (d+k+1)r\Big)$ be the $d+k+1$ values returned by program f at points $x+r, \ldots, x+(d+k+1)r$.

Theorem 5. *The system of equations (6) is satisfied if and only if the number of $f(x+ir)$ that are wrong is equal to 0 or greater than k.*

$$\begin{cases} f(x+r) &= \displaystyle\sum_{i=1}^{d+1} (-1)^{i+1} C_{d+1}^{i} f\Big(x + (i+1)r\Big), \\ \quad\vdots & \qquad\qquad \vdots \\ f(x+kr) &= \displaystyle\sum_{i=1}^{d+1} (-1)^{i+1} C_{d+1}^{i} f\Big(x + (i+k)r\Big). \end{cases} \tag{6}$$

The satisfaction of (6) can be checked easily since the algorithm needs only to verify that the values of $P(x+ir)$ are equal, by querying directly the program f and by reconstructing its value by RSR. This technique allows us to detect k errors from $d+k+1$ values of the polynomial. The detection is very simple given the structure of the control matrix.

5.1. A self-correction algorithm based on early error detection

We first study the characteristics of Algorithm (2) and then compare them with the algorithm based on a simple vote. The comparison is carried out by considering the number of queries necessitated by the algorithm to reach a given failure rate, and the error probability of the defective program that the algorithm tolerates.

Algorithm 2. *Early error-detecting, self-correcting program(f)*

$r \leftarrow$ random element from F
While $P(x+r) \neq \sum_{i=1}^{d+1}(-1)^{i+1}C_{d+1}^{i} f(x+(i+1)r)$ **do:**
    $r \leftarrow$ random element from F
result $\leftarrow \sum_{i=1}^{d+1}(-1)^{i+1}C_{d+1}^{i} f(x+ir)$
end

5.2. Characterizing the early detection algorithm

The probability that the final result is correct at round n is equal to $p_{\text{stop}} * p_{\text{correct}}$, where p_{stop} is the probability that the algorithm stops after round n and p_{correct} is the probability that the final result is correct.

1. The algorithm stops before round n with probability:

$$p_{\text{stop}} = \sum_{i=0}^{n-1}(1-p)^{d+2}\left(1-(1-p)^{d+2}\right)^{i},$$
$$p_{\text{stop}} = 1-\left(1-(1-p)^{d+2}\right)^{n}. \tag{7}$$

2. The probability that the final result is correct is computed as follows:

$$p_{\text{correct}} = \frac{\text{Probability that all } f(x+ir) \text{ are correct}}{\text{Prob that 0 or more than 2 values are wrong from (d+2) } f(x+ir)},$$
$$p_{\text{correct}} = \frac{(1-p)^{d+2}}{1-(d+2)p(1-p)^{d+1}}. \tag{8}$$

5.3. Comparison with the simple voting algorithm

Both algorithms (1) and (2), allow the self-correction of programs that evaluate polynomials of degree d. The degree of the polynomial to be corrected is the only prerequisite needed by these algorithms. We compare them on the basis of two criteria: 1) the initial error probability of the program to be corrected that can be tolerated by the algorithms, 2) the number of queries necessary to reach a given error probability.

Gain in tolerated error probability: To apply the self-correction algorithm (1) based on vote, it is necessary that the values reconstructed by RSR have an error probability less than $1/2$. Since one value value is reconstructed by querying the defective program on $d+1$ points, the error probability of the defective program has to verify: $1-(1-p)^{d+1} < \frac{1}{2}$ or $p < \frac{1}{2(d+1)}$. Self-correction with early detection of errors is less restricting (see Figure 1).

Gain in the number of queries to achieve the same error probability In Section 2, we have shown that the majority vote has an error probability β, with an initial error probability $\frac{p'}{d+1}$ and with $(d+1)\frac{2(1-p')\ln\frac{1}{\beta}}{(p'-1/2)^2}$ queries. For the same initial error probability $p = \frac{p'}{d+1}$, we compare the number of queries (i.e., $(d+2)*n$) necessary for Algorithm (2) to reach an error probability

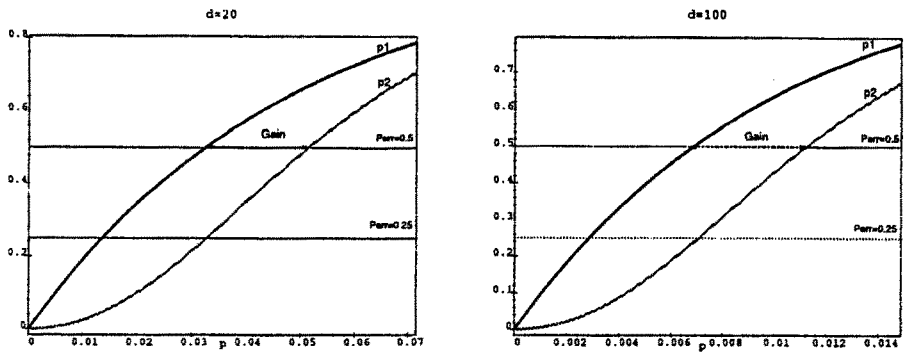

Figure 1. The curves represent the gain in tolerance

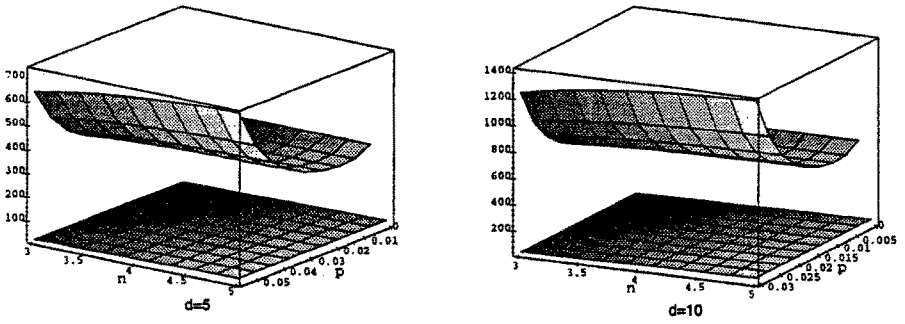

Figure 2. Number of queries necessary for the two algorithms to reach the same error probability. The parameters are the initial error probability p of the defective program and the number n of rounds of the early detection algorithm

of $1 - p_{\text{correct}} * p_{\text{stop}}$ and the number of queries $\left(\text{i.e., } (d+1)\frac{2(1-p')\ln\frac{1}{1-p_{\text{correct}}*p_{\text{stop}}}}{(p'-1/2)^2}\right)$ necessary for Algorithm (1) to reach the same error probability.

The early detection algorithm self-corrects programs with an error probability greater than the error probability accepted by the algorithm based on the majority vote (see Figure 1). Furthermore, our algorithm is much simpler than the ones based on codeword error correction since it only performs random self-reducibility computations and comparisons. Finally, the major advantage of our algorithm is that it requires significantly less queries than the algorithm based on the majority vote (Figure 2).

6. Conclusion

In this paper, we gave a lower bound on the order of RSR of polynomials and introduced new RSR formulas. Furthermore, we have presented an efficient algorithm for self-correcting polynomials.

References

[1] Blum, M. et al. *Self-testing and self-correcting programs, with applications to numerical programs.* In: "Proceedings of the 22th ACM STOC'90".

[2] Lipton, R. J. *New directions in testing.* In: "Distributed Computing and Cryptography", volume 2 of DIMACS, ACM AMS, 1991.

[3] Noubir, G. and Nussbaumer, H. J. *Self-correcting polynomials.* Swiss Federal Insitute of Technology in Lausanne, CS–TR–95/151.

[4] Nussbaumer, H. J. *Fast Fourier transform and convolution algorithms.* Springer Series in Information Science, Springer-Verlag, 1982.

[5] Rubinfeld, R. et al. *Self-testing/correcting for polynomials and for approximate functions.* In: "Proceedings of the 23rd ACM STOC'91".

[6] Sudan, M. *Efficient checking of polynomials and proofs and the hardness of approximation problems.* Ph.D. Thesis, University of California at Berkeley, 1992.

Received: October 16, 1995
Revised version: November 27, 1995

Swiss Federal Institute of Technology in Lausanne
(EPFL)
Computer Science Department
CH–1015
Switzerland
E-mail: noubir@litsun.epfl.ch