Determining Equivalence of Expressions in Random Polynomial Time

Extended Abstract

by

Gaston H. Gonnet

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

We devise several procedures based on signatures (or hashing functions) to determine equivalence of expressions in Random Polynomial Time (also called Probabilistic Polynomial Time) (RPT). These procedures return as result: *"equivalent"* or *"not-equivalent"*. The result "not-equivalent" is always correct, while the result "equivalent" is correct with probability at least $1-\varepsilon$. This probability depends on a random number generator and is *independent* of the problem being solved. In all our procedures, the value ε can be made arbitrarily small. This method works for determining equivalence over an important class of functions as well as answering other questions like linearity, polynomial dependence, squareness, independence, etc.

The general scheme for all these algorithms is to use a basic heuristic "*test*" several times. I.e.

```
solve( problem, ε )
repeat
        select suitable characteristic p randomly;
        assign random hash values to all variables;
        t := test( problem, p );
        if t = "not-equivalent" then return( t )
        until
                probability of cumulative failures ≤ ε;
return( "equivalent" );
```

It is assumed that "*test*" gives a wrong answer with probability δ, where δ remains bounded below 1. For all our "*test*" procedures, $\delta \leq 1/2$.

In what follows we will describe the "*test*" part of the different procedures, it is aways assumed that these are used in the above context.

Without loss of generality we will assume that equivalence of expressions ($A \equiv B$) can be transformed into testing for 0 ($A - B \equiv 0$).

1. Expressions involving integers, variables and arithmetic operations

A signature of an expression, $s(expr)$, satisfies the following properties:

$s(x) = x \mod p$ if x is an integer;

$s(x) = h(x) \mod p$ if x is a variable, and $h(x)$ is a hashing function on variables which returns an arbitrary integer;

$s(a+b) = (s(a)+s(b)) \mod p$
idem for $-$, $\times$ and $/$.

The above signatures are said to be computed with *characteristic p*, which is normally a prime number.

It is straightforward to check that an expression equivalent to 0 will always return the value 0. Probabilistic testing of an expression being identical to 0 is achieved by testing whether its signature is 0. Of course, non-zero expressions can also have a zero signature, but as we will show, this can be made to happen with low probability.

Let $T(f)$ be the complexity of evaluating the expression f, or what is sometimes called the *programme complexity* of f. $T(f)$ counts the number of arithmetic operations needed to evaluate the expression in any particular field where every operation counts as one. For example

$$T((x^8z^2+w)^4) = 7$$

The evaluation of one signature $s(f)$ can be done in $O(T(f)\log p)$ operations as operations in finite fields can be done in at most $O(\log p)$ basic operations.

A typical example of this type of problem is:

$$\frac{(x^2-1)^{4096}}{x-1}-(x+1)^{4095}-\prod_{i=0}^{12}((x-1)^{2^i}+1) \equiv 0 \quad ?$$

1.1. Bounding the probability of error

To bound the probability of an erroneous answer from the "*test*" procedure, or more precisely $Pr\{s(f)=0 \mid f\neq 0\}$, we have the following approach:

Every expression could be expanded into a quotient of two polynomials, and if d is the total degree of the polynomial in the numerator, then the probability of error is $\delta\leq d/p$. A proof of this theorem can be found in [Schwartz,80].

Since the total degree in the expression cannot be higher than $2^{T_{\times/}(f)}$, where $T_{\times/}(f)$ is the number of multiplications and divisions used to compute f, selecting $p \geq 2^{T_{\times/}(f)+1}$ is enough to guarantee $\delta\leq 1/2$.

Lemma 2: The above bound is tight.
E.g. for polynomials $q(x)=x^{2^k}-1$ when $2^k \mid \phi(p)$. ($\phi(p)$ is Euler's totient function and counts the number of integers less than p which are relatively prime to p.) This polynomial requires k multiplications to evaluate and for such a value p, there are 2^k roots of unity. I.e. 2^k values for which $q(x)=0$.

It is convenient to define

$$R(f) = O(T(f)\log p \log_\delta \varepsilon),$$

the complexity of evaluating all the signatures needed for one instance of equivalence testing. Using the above bound, and for optimal selection of p which happens when $\delta\approx\varepsilon$ the total complexity can be reduced to

$$R(f) = O(T(f)\,(T_{\times/}(f)+\log 1/\varepsilon))$$

To translate to bit complexities (or if we have to use multiple precision to represent signatures) we have to multiply the above values by $M(\log p)$ where $M(n)$ is the complexity of multiplying two n-digit/bit numbers.

At least two technicalities have to be taken into consideration:

Division by 0: In this situation we recursively test for the divisor being identical to zero. (E.g. $1/(x^2-1+(1-x)(x+1))$) If the divisor is a real zero, the expression cannot be computed and equality cannot be tested; else we stop the computation and restart with a new set of random assignments for the indeterminates and the characteristic. In the worst case this can happen with cumulative probability δ. This will increase the total running time by at most $1/(1-\delta)\leq 2$.

Selection of characteristic p: It is always possible to find in RPT a prime number of the form $p=nk+1$ for a given k. This is achieved by trying with consecutive values of n with a probabilistic primality testing algorithm [Solovay et al.,77], [Rabin,76]. The expected number of trials is less than $\ln p$ for any $k\neq 1$. It is also desirable to find a prime number greater than some bound P. For large

enough P, the expected complexity of finding such a prime number is $O((\log P)^2 \log\varepsilon)$. By finding a prime in this way, and assuming that we know the complete factorization of k (as this is the case for our algorithms) we can easily find primitive roots of p.

Alternatively we can have pre-computed large primes with known primitive roots etc. The point made above is that pre-computed primes are not really needed.

With some variations and/or restrictions this basic approach is described, or can be inferred, from [Martin,71], [Schwartz,80], [Ibarra,83] and [Welsh,83]. Heintz et-al. [80] prove the existence (but this is a non-constructive proof) of short sequences of assignments to prove polynomial equivalence. One of the applications of this basic test is the determination of singularity and rank of a matrix with multivariate polynomial entries [Edmonds,67], [Ibarra et-al.,81].

1.2. Complex arithmetic

Complex expressions are readily handled by selecting primes p for which $i=\sqrt{-1}$ can be represented, or equivalently $p=4k+1$. For example, for $p=13$, $s(i)=5$, as $5^2=-1 \ mod\ 13$.

Theorem 3: Equivalence testing using complex arithmetic can be done in RPT.

1.3. Non-prime characteristics

For non-prime characteristics, it is possible to compute signatures as described above. Divisions will fail with a probability higher than $1/p$, though. If a division fails, we may try again selecting different signatures and/or characteristic. Let $T_f(f)$ be the number of divisions with different divisors used to evaluate f, then, assuming random signatures, the probability of success is

$$\left(\frac{\phi(p)}{p}\right)^{T_f(f)}$$

The expected complexity of computing signatures becomes

$$R(f)\left(\frac{p}{\phi(p)}\right)^{T_f(f)}.$$

We now describe several extensions to the applicability of signatures and the way these impact the basic algorithms.

2. Roots and rational exponents

It is straightforward to compute the signature of an expression powered to a rational exponent:

$$s(x^{a/b}) \ = \ (s(x)^a)^{1/b}$$

If $1/b$ can be represented *mod* $\phi(p)$, direct application of Fermat's little result is sufficient. If, however, $gcd(b,\phi(p))\neq 1$ then it is not always possible to compute roots in the finite field. Let $b=b_1^{a_1}b_2^{a_2}\cdots$ be the prime number factorization of b. Then finding each individual b_i^{th} root is equivalent to finding the b^{th} root. For each $b_i\neq 2$ we can simply select a number p such that b_i does not divide $\phi(p)$. But since $\phi(p)$ is always even, the difficulty of computing square roots cannot be removed. For any characteristic p, only half of the numbers have square roots.

Two important special cases can be solved for the square root problem. Square roots of variables can be successfully computed by forcing the signature of the variable to be selected from the residues *mod* p. (Notice that this is really equivalent to substituting a variable for a new variable which is equal to its square root.) Square roots of integers, ($\sqrt{j} \ mod\ p$) and as an immediate consequence of rationals, can be guaranteed to exist if we select p such that $4j$ divides $\phi(p)$, or $p=4nj+1$. The sign of the radicals has to be consistent, i.e. $s(\sqrt{6})=s(\sqrt{2})s(\sqrt{3})$, which can be easily guaranteed by computing the independent factors of all integer square roots which involve computing several integer gcds.

Lemma 4: Let $T_\surd(f)$ be the number of square roots computed on different arguments (excluding rationals and indeterminates) while evaluating f, then the probability of being able to compute a successful signature is $2^{-T_\surd(f)}$. Therefore the testing becomes exponential in $T_\surd(f)$.

3. Exponentials and trigonometric functions

3.1. Exponentials

Let $s(e)$ be an arbitrary primitive root of p, where e denotes the base of the natural logarithms.

Lemma 5: $s(e)^i \bmod p$ is independent of any polynomial $q(i) \bmod p$ with $deg(q(x)) < p-1$. The proof follows from taking enough divided differences.

Consequently the signature of e^i can be computed as $s(e^i) = s(e)^i \bmod p$. This definition satisfies the usual rules of the exponential function and will be algebraically independent of any polynomial.

To handle a general exponent, we have to be aware that the exponents should be computed $\bmod \phi(p)$. For this purpose we will define levels of signatures. In short, if a level uses the number p for its signatures the next level, the one used in exponents, is associated with $\phi(p)$. For any $s(e^x)$ computed at level j, e is computed at level j and x is computed at level $j+1$.

$$s_j(e^x) = \left(s_j(e)\right)^{s_{j+1}(x)} \bmod p$$

Furthermore, for levels greater than 1 we can define the signature of the logarithm function: $s_j(\ln(x)) = \gamma$ implies that $s_{j-1}(e)^\gamma = s_{j-1}(x)$.

The exponential can be generalized to any other expression a^x. By forcing $s(a)$ to be a primitive root $\bmod p$ we can guarantee a behaviour similar to that of e^x. Let $T_{base}(f)$ be the number of different expressions a where a^x appears in the computation, then the signature computation will succeed with probability

$$\left(\frac{\phi(\phi(p))}{p}\right)^{T_{base}(f)}$$

Of course this could be relaxed to just checking that $s(a)$ is not a "poor" choice (a primitive root is the best choice). E.g. if $s(a)=0$ or $s(a)=1$, the signature of $s(a^x)$ is non-discriminating on x.

3.2. Trigonometric functions

Since it is possible to represent e^x and i, it appears that it is possible to use

$$\sin(x) = \frac{e^{ix}-e^{-ix}}{2i}$$

and compute $\sin(x)$ in terms of exponentials. Unfortunately this is not possible.

Lemma 5: i cannot be represented at two consecutive levels.

Even if this was possible, π, e and i should satisfy the equation $e^{i\pi}+1=0$; but this implies, for example,

$$\cos(x) = \frac{(-1)^{x/\pi}+(-1)^{-x/\pi}}{2}$$

and consequently the signature of $\cos(x)$ is trivial. It appears that the main difficulty is representing $(-1)^{1/\pi} \bmod p$. Trying a more fundamental approach, we can test whether any value from $0 \cdots p-1$ assigned to $(-1)^{1/\pi}$ properly defines $\sin(x)$ and $\cos(x)$. For this purpose we will define a correct trigonometric function as one for which:

$$\begin{aligned}
\sin(a+b) &= \sin(a)\cos(b)+\cos(a)\sin(b) \\
\sin^2(x)+\cos^2(x) &= 1 \\
\sin(x) &= \cos(x-\pi/2) \\
\sin(0) &= \sin(\pi) = 0 \\
\cos(0) &= \cos(\pi) = 1
\end{aligned}$$

From these, all the usual trigonometric relations and special argument relations can be derived. The above conditions can be satisfied if we let $s_{j+1}(\pi) = \phi(p)/2$ at level $j+1$, let $\phi(p)$ be divisible by $4F$ and let g be a primitive root of p. Then at level j we define

$$s_j(\sin(x)) = \frac{g^{s_{j+1}(x)} - g^{-s_{j+1}(x)}}{2i} \quad mod\ p$$

$$s_j(\cos(x)) = \frac{g^{s_{j+1}(x)} + g^{-s_{j+1}(x)}}{2} \quad mod\ p$$

and

$$s_j(i) = g^{-\phi(p)/4}$$

and all other trigonometrics can be expressed in terms of $\sin(x)$ and $\cos(x)$. F is the product of all Fermat primes which appear in the computation. This requirement is to allow the successful checking of identities involving special angles $(\pi/3, \pi/5, \pi/17, \cdots)$ which can be expressed in terms of arithmetic and square root operations. The primitive root g should be different from $s(e)$ and furthermore, to prevent simple polynomial relations between $\sin(x)$ and e^x, $\log_g s(e)\ mod\ p = O(p)$ and $\log_{s(e)} g\ mod\ p = O(p)$. Note that both functions have symmetries which will imply that the effective range of $s(\sin(x))$ and $s(\cos(x))$ compared to the effective range of $s(x)$ is reduced by half; as bad as for multiplication/ division.

3.3. A new class of functions

We define SIG to be the class of functions for which we can compute their signature and hence determine equivalence in RPT. Then

$$Z \cup \{ variables, i \} \subset \text{SIG}$$

if $x,y \in$ SIG then

$$\{ x+y, x-y, xy, x/y, \sqrt{Z}, \sqrt{variables} \} \subset \text{SIG}$$

Provided that we can compute the signatures $s_j(y)$ at level j, $j>1$ (this will mean, among other restrictions, that we do not have explicit or implicit divisions by 2 as these will be equivalent to computing square roots):

$$\{ x^y, e^y, \sin(y), \cos(y), \cdots \} \subset \text{SIG}$$

4. Implicit equations and solution of equations

It is possible to extend the present scheme to answer queries of the type: is $f(x,y,z)=0$, where $g(z)=0$. For example, is

$$\cos^2 z(\cos(z+y)\cos(z-y)+3) - z(\cos^2 y + 2 + z) = 0$$

when

$$\sin^2 z + z = 1 \quad ?$$

For this situation, the signature of z will not be an arbitrary integer, but will be such that satisfies $s(g(z)) = 0$.

Finding the values for $s(z)$ such that $s(g(z))=0$ may require work proportional to p, as the only method available may be exhaustive search. For a prime number p this may be the end of the story, but for composite p, in particular if $p=t^k$, we can use a method similar to Hensel lifting where we solve the equation for $g(z)$ for p, then based on the solution for p we compute the solution for p^2, etc. E.g.

$$g(z) = z^3 - \frac{5z}{z+3} - \sqrt{z+1} = 0$$

then for $p=17$, if $s(z)=8$ then $s(g(z))=0$;
for $p=17^2=289$, $s(z) = 76 = 17\times 4+8$,
for $p=17^3 = 4913$, $s(z) = 2677 = 289\times 9+76$, etc.

The total time to compute the signature of the solution by lifting is $O(k\,t\,T(g(z)))$.

Equation solving is an interesting topic in itself, as there are several questions which can be answered by signatures.

E.g. suppose that $f(x,y)=0$ and we are interested in the implicit function $x=f^*(y)$, where $f(f^*(y),y)$ is identically zero. By finding the pairs of signatures for x and y which satisfy $f(x,y)=0$ we can, in RPT

(i) determine whether $x=a_0+a_1y$ and determine a_0 and a_1 if these are rationals;

(ii) determine whether $x=f^*(y)$ is in SIG or not;

(iii) determine whether the equation is infeasible/indeterminate.

E.g., let

$$f(x,y) = \frac{x^2y}{xy+3} - \frac{5x-y}{(x+1)^{1/3}-y} = 0$$

then the signatures of x and y which give $s(f(x,y))=0 \ mod\ 11$ are

$s(x)$	0	1	2	6	6	7	10	10
$s(y)$	0	3	2	2	7	4	1	9

Then neither $x=f_1(y)$ nor $y=f_2(x)$ are in SIG as there are missing signature values for both x and y.

This extends our class SIG by adding implicit equations or side relations. I.e. if $x,y \in$ SIG then $(x,y\equiv 0) \in$ SIG, where the notation $(x,y\equiv 0)$ means x subject to the condition $y\equiv 0$. If each side relation can be associated with a unique indeterminate on which we will do the lifting, the testing can be done in $O(ktR(f))$. If we do not have enough indeterminates, the process will become exponential in the number of side relations without an "assignable" indeterminate.

4.1. Retrieval of rational numbers.

In some cases we are interested in obtaining a rational (or integer) number from its signature. This process is the inverse of computing a signature and is also called *lifting*. Let $s(m/n)=r$, then we can find m/n in RPT using an algorithm almost identical to the algorithm for finding inverses in finite fields:

$$m_{-1} = p, \quad m_0 = r, \quad m_{i+1} = m_{i-1} - c_{i+1}m_i$$
$$n_{-1} = 0, \quad n_0 = 1, \quad n_{i+1} = n_{i-1} + c_{i+1}n_i$$
$$c_i = \lfloor m_{i-2}/m_{i-1} \rfloor$$

then $\dfrac{(-1)^i m_i}{n_i} \equiv r \ \ mod\ p$ and these numbers are "small" ($m_i n_i \leq 2p$).

To retrieve the rational we select $p >> mn$ which guarantees that m/n will appear as one of the terms in the m_i/n_i sequence. The common fraction for several values of p will be m/n.

5. Various tests

5.1. Linearity and polynomial dependence test

Going back to the original procedure, we can determine in RPT whether $f(x,y,...)$ is

(i) linear in one variable

(ii) polynomial, with degree less or equal to k, in any of the variables and determine its degree.

The first test is a special case of the second. The latter testing deserves a quick explanation. The basic function is to determine whether $f(x,y,...)$ is of degree k or less. To do this we generate $k+2$ different random signatures for x while maintaining the signatures for the other variables fixed. For the first $k+1$ signatures we compute $s(f(x,y,...))$ and interpolate a Lagrange k-degree polynomial. Finally we use this polynomial to check for the $k+2^{nd}$ point.

To find the exact degree, given an upper bound, we can do binary search in the degree, using a total of $O(R(f)k^2 \log k)$ units of time.

5.2. Independence test

Independence with respect to a given variable, say x, can be tested by computing the signature of $s(f(x,...))$ for different hashing values of x, but keeping the same assignments for the other variables. If the signatures differ, then $f(x,...)$ is not independent of x. To achieve an answer with probability ε we need $O(R(f))$ basic operations.

To determine whether a function is a constant, we determine independence with respect to all its indeterminates. Notice that a constant value may have different signatures for different characteristics; e.g. $3/7 \equiv 44 \bmod 61$, $3/7 \equiv 41 \bmod 71$, similarly for $\sqrt{5}$, e^2, etc.

5.3. Perfect-square testing

If an expression is the perfect square of an expression in SIG then its signature will be a residue *mod* p. For prime p, half of the numbers are residues and the other half are non-residues. The testing for residues is equivalent to evaluating the Legendre's symbol, which can be done in $O(\log p)$ operations. The total complexity of testing for perfect squareness with error probability ε is $O(R(f))$.

6. Conclusions and open problems

We could not resist the temptation for implementing a prototype package for the Maple system [Char,83]. The conclusion is that signatures work remarkably well, and that the probability bounds proven here are, in practice, very pessimistic. This type of equivalence checking, not only works for problems for which the conventional approach is hopeless, but for moderate size problems is orders of magnitude faster.

6.1. Open problems

(a) Is it possible to define $\log(x)$ at the basic level? Similarly for arc-trigonometric functions?

(b) Can we break the barrier for computing signatures of square roots?

(c) Gaussian signatures? Complex modular arithmetic?

(d) A new arithmetic model altogether.

(e) Is it possible to define the signature of other functions $\int f(x)\,dx$, $f'(x)$, $\sum f(i)$ and operators in general? i.e. extend SIG?

(f) Is it feasible to operate with sets of signatures, and hence handle some multiple-valued functions, or abs() ?

7. References

[1] Edmonds, J.: Systems of Distinct Representatives and Linear Algebra; Journal of Research of the NBS, 71B(4):241-245, (Dec 1967).

[2] Hardy, G.H. and Wright, E.M.: *The Theory of Numbers*; Oxford Clarendon Press, Fourth Ed, (1975)

[3] Heintz, J. and Schnorr, C.P.: Testing Polynomials which are easy to compute; Proceedings of An International Symposium Held in Honour of Ernst Specker; Monographie 30 de l'enseignement Mathematique; 237-254, (Feb 1980)

[4] Ibarra, O.H., Moran, S. and Rosier, L.E.: Probabilistic Algorithms and Straight-Line Programs for Some Rank Decision Problems; Inf Proc Letters, 12(5):227-232, (Oct 1981)

[5] Ibarra, O.H. and Moran, S.: Probabilistic Algorithms for Deciding Equivalence of Straight-Line Programs; J.ACM, 30(1):217-228, (Jan 1983)

[6] Martin, W.A.: Determining the Equivalence of Algebraic Expressions by Hash Coding; J.ACM, 18(4):549-558, (Oct 1971)

[7] Moses, J.: Algebraic Simplification: A Guide for the Perplexed; C.ACM, 14(8):527-537, (Aug 1971).

[8] Rabin, M.O.: Probabilistic Algorithms; in Algorithms and Complexity, Academic Press, New York, (1976).

[9] Rabin, M.O.: Probabilistic Algorithms in Finite Fields; Siam J Computing, 9(2):273-280, (May 1980)

[10] Schwartz, J.T.: Fast Probabilistic Algorithms for Verification of Polynomial Identities; J.ACM, 27(4):701-717, (Oct 1980)

[11] Solovay, R. and Strassen, V.: A Fast Monte-Carlo Test for Primality; SIAM J Computing; 6:84-85, (1977).

[12] Welsh, D.J.A.: Randomised Algorithms; Discrete Mathematics, 133-145, (1983).

[13] Zippel, R.: Probabilistic Algorithms for Sparse Polynomials; Proceedings Eurosam 79, Lecture Notes in CS 72:216-226, (1979)