Deterministic Primality Testing in Polynomial Time

A. Gabriel W. Daleson

December 11, 2006

Abstract

The new Agrawal-Kayal-Saxena (AKS) algorithm determines whether a given number is prime or composite in polynomial time, but, unlike the previous algorithms developed by Fermat, Miller, and Rabin, the AKS test is deterministic. The new test is not without its disadvantages, however; the older, error-prone Miller-Rabin test is accurate enough for nearly all numbers, and both experiment and analysis show that the AKS algorithm is, at present, still too slow to replace its predecessor.

1 Introduction

Primality 1. A natural number N is *prime* if and only if there exist no products of the form $ab = N$ where both a and b are not equal to 1 or N.

The definition of primality is one of the trickiest in mathematics.

Firstly, the definition is a negative one; to prove that N is prime, we must show not that something happens, but that something does *not* happen. In most circumstances, this means exhaustive search; granted, we do get to stop whenever we find the first nontrivial factorization, but, in the case where N is indeed prime, we only find this after examining all possibilities, which can take a very long time.

For our purposes, though, it's not the negative aspect of the definition that causes the problem, but the search. Algorithmically speaking, searches are awful; only in the very rarest cases can we arrange things to make searches

quick. What we would prefer is a more functional definition, where we can apply some computation to N that gives us the answer, but, at least at this early stage, we lack such things.

This poses a rather serious problem, in that current cryptographic algorithms such as RSA require large primes; currently, software such as the GNU Privacy Guard can require primes between 2^{2047} and $2^{2048} - 1$. We therefore need a quick, accurate way to test large numbers for primality.

2 Fermat's Little Theorem & Test

Our effort begins with the following theorem:

Fermat's Little Theorem 1. *For any $a \in \mathbb{Z}_p$, $a^p = a$.*

The proof proceeds via induction on a. In the cases $a = 0$ and $a = 1$, the theorem is plainly true. Supposing it holds for some $k \in \mathbb{Z}_p$ — that is, that $k^p = k$ — we can consider $(k+1)^p$. Now, the binomial theorem tells us that this looks like

$$k^p + 1 + \sum_{j=1}^{p-1} \binom{p}{j} k^j$$

but we know that $\binom{p}{j} = \frac{p!}{(p-j)!j!}$ will always be a multiple of p, as $0 < j < p$, so $p - j < p$ and $j < p$; as p is prime, this means that neither $(p-j)!$ nor $j!$ will ever contain factors of p. Therefore, every term in the summation is 0 in $\mathbb{Z}_p$, giving us $k^p + 1$, which, by the induction hypothesis, is $k + 1$. We therefore have proven that, given a $k \in \mathbb{Z}_p$ such that $k^p = k$, it follows that $(k+1)^p = k + 1$; adding this to our base case, the induction is complete. ■

We might hope (albeit in vain) that the converse is true; that is, that if $a^N = a$ for some $a \in \mathbb{Z}_N$, that N is prime. This is our first primality test.

The Fermat Test 1. *If there exists an a, not 0 or 1, such that $a^N = a$ in $\mathbb{Z}_N$, then N is prime.*

Closely related is the Chinese Hypothesis, which states that N is prime if and only if $N|2^N - 2$ (or, if you prefer, that $N|2^{N-1} - 1$ for $N \neq 2$).

Both the Fermat Test and the Chinese Hypothesis are, however, false; the smallest number for which the Chinese Hypothesis fails is $341 = 11 \times 31$. This is, fortunately for us, rather easy to see, as $341 \times 3 = 1023$, which is

easily recognized as $2^{10} - 1$. From this, it follows that $2^{10} = 1$ in $\mathbb{Z}_{341}$, and so $(2^{10})^{34} = 2^{340} = 1$ in $\mathbb{Z}_{341}$.

In fact, it can fail for any a; for instance, setting $N = 91$ and looking at $3^N - 3$, note that $91 \times 8 = 728$, one less than $729 = 3^6$.

3 Psuedoprimes

Given, then, that the Fermat Test has these holes, the logical way to proceed is to investigate them.

k-pseudoprimes 1. A composite number N is said to be *k-pseudoprime* or a *Fermat k-liar* if and only if $k^N = k \pmod{N}$.

Now, using this language of k-pseudoprimes, we can at least amend things a bit and make them correct, if not yet useful. As it stands, the old version of the Fermat Test only gives correct results when it proves a number composite.

An Amended Fermat Test 2. *If there exists an k, not 0 or 1, such that $k^N = k$ in $\mathbb{Z}_N$, then N is either prime or k-pseudoprime.*

Now, we can outmaneuver things a bit by varying k, but then how many different ks do we have to use in the Fermat Test to prove a number prime? In fact, might it not be enough if we test every k? We can at least make the following definition:

Absolute Pseudoprimes 1. A composite number N is said to be an *absolute pseudoprime* or *Carmichael number* if and only if N is k-pseudoprime for all k, or, more explicitly, if $k^N = k \pmod{N}$ for all k.

The usual way we identify absolute pseudoprimes is via Korselt's Criterion [Car10].

Korselt's Criterion 1. *N is an absolute pseudoprime if and only if*

i. N is composite.

ii. N is squarefree.

iii. for all prime p, if $p|N$, then $p - 1|N - 1$.

The proof of Korselt's Criterion is quite simple. Suppose N satisifes the criterion; then, as N is squarefree, we know that in order for N to divide $k^N - k$, all we require is that $p|k^N - k$ for all $p|N$. Fortunately, Fermat's Little Theorem tells us that, when $k \neq 0$, $k^{p-1} = 1 \pmod{p}$ for all k, and, as $p-1|N-1$, we know that $k^{N-1} = 1 \pmod{p}$; given that $p|k^{N-1} - 1$ for all $p|N$, it follows by taking the least common multiple of all such p (again taking advantage of the fact that N is squarefree) that $N|k^{N-1} - 1$, or that $k^{N-1} = 1 \pmod{n}$; multiplying by k to add the trivial case $k = 0$ tells us that Korselt's Criterion implies absolute pseudoprimality.

Conversely, let N be an absolute pseudoprime; we take it as true that $N|k^N - k$ for all k. First, we prove that n must be squarefree; if N is not squarefree, then we can find some factor of N with the form c^2. Note, then, that, as $c^2|N$ and $N|c^N - c$, we have $c^2|c^N - c$. Note that, as $c^2|c^N$ trivially, this forces $c^2|c$, which is impossible. Thus, it must be that N is squarefree, as we expected.

Now, it remains to prove that if $p|N$, then $p-1|N-1$; for this, we turn to some simple theory of the finite group $\mathbb{Z}_p^\times$. We know that this group is cyclic, and therefore we can chooze a generator a, which will have group order $p-1$. We have $p|N$, and, by hypothesis, $N|a^N - a$, so $p|a^N - a$. We know that $p \not| a$, so therefore, we can derive that $p|a^{N-1} - 1$. It is clear from this that $a^{N-1} = 1 \pmod{p}$, and it follows from this that $N - 1$ must be a multiple of $p - 1$, as desired. Combining this with the result of the previous paragraph gives us the final result, Korselt's Criterion for absolute pseudoprimality. $\square$

Note that, apart from the first condition, prime numbers trivially satisfy Korselt's Criterion.

It's worth talking a bit about the nature of this theorem. Fermat's Little Theorem, in and of itself, is, in the context of group theory, a very simple corollary of Lagrange's Theorem on the order of finite groups. What Korselt's Criterion tells us is that the mathematics of primality is, in some sense, not quite aligned with this group theory; Fermat's Little Theorem works in too many cases, and, when we have N squarefree and every prime divisor p of n satisfies $p-1|N-1$, then, when we calculate a^{N-1}, we're really taking $(a^{p-1})^c$, and, as $a^{p-1} = 1 \pmod{p}$, Lagrange's and Fermat's Little Theorem apply, even though for our current purposes, we'd rather they not.

Also, rather oddly, Korselt's Criterion came along first; while he proved this necessary and suficient condition for absolute pseudoprimality, he never was actually able to get his hands on such a number. Carmichael did this

one year later.

The first thing Carmichael proved was that absolute pseudoprimes have at least three prime factors. Supposing N is an absolute pseudoprime of the form pq, we get that $p-1|pq-1$, so $\frac{pq-1}{p-1}$ must be an integer. We can recast the numerator $pq-1$ as $(p-1)q+q-1$, which tells us that $\frac{pq-1}{p-1} = q + \frac{q-1}{p-1}$. The question now is, do there exist primes p and q so that $p-1|q-1$ and $q-1|p-1$? This is clearly not possible; we would have to have $p-1 = q-1$, and thus that $p = q$, making $n = pq$ a perfect square and obviously not squarefree.

We can, though, get our hands on absolute pseudoprimes with three distinct prime factors; suppose N is an absolute pseudoprime, and that it is the product of three distinct primes; $N = p_1p_2p_3$. By Korselt's Criterion, the quantity $\frac{p_1p_2p_3-1}{p_1-1}$ must be an integer. Cleverly rewriting $p_1p_2p_3 - 1$ as $(p_1-1)p_2p_3 + p_2p_3 - 1$ and dividing, we get that $\frac{p_1p_2p_3-1}{p_1-1} = p_2p_3 + \frac{p_2p_3-1}{p_1-1}$; in order for this to be an integer, we require that $p_1 - 1|p_2p_3 - 1$. Note, too, that the analogous conditions apply; we must also have $p_2 - 1|p_1p_3 - 1$ and $p_3 - 1|p_1p_2 - 1$.

We can easily put together a few valid absolute pseudoprimes using these simple conditions; Carmichael's early lists included $3 \times 11 \times 17 = 561$ (the smallest absolute pseudoprime), $5 \times 13 \times 17 = 1105$, $7 \times 13 \times 31 = 2821$, and $7 \times 31 \times 73 = 15841$. Simply as demonstration, in the case of 561, note that we get $3 \times 11 = 33$, $3 \times 17 = 51$, and $11 \times 17 = 187$, and in each of these cases we get $17 - 1 = 16|32 = 33 - 1$, $11 - 1 = 10|50 = 51 - 1$, and $3 - 1 = 2|186 = 187 - 1$. Note that nowhere did we require that absolute pseudoprimes have only three prime factors; one can easily extend Carmichael's original calculation to cases with arbitrarily many prime factors, though of course the arithmetic does get to be a little much.

So, now we find that, at least in the context of our nice, simple idea of the Fermat Test, we have to amend it a bit to work in what we find is now less a nice, clear plain and is more of a minefield. While what we have now is not mathematically perfect, it is still useful, and we should proceed with discussing its algorithmic aspects.

4 Complexity Analysis & Fast Modular Exponentiation

First, we have to discuss a few things regarding algorithms in general. The major concern at this point is, were we to try and write a program for the (Amended) Fermat Test, a bit of cleverness would be required.

Consider, as stated above, that N is on the order of 2^{2048}, and we need to calculate $a^N \bmod N$. If we should dare apply our naïve, inductive definition of powers to this calculation — namely, that $a^0 = 1$ and that $a^{x+1} = a \times a^x$ — we would find ourselves having to perform $2^{2048} - 1$ multiplications.

On an abstract level, this is no problem whatsoever; we can certainly *imagine* carrying out this many multiplications.

But.

Modern computers, being generous, can perform around 2 billion multiplications and reductions per second. Relative to a number like 2^{2048} — which we can estimate as 10^{692} — we're in a speed bracket well below Windows 95, rush hour traffic, and glaciers. We're talking about calculations that take 10^{600} years or so. NASA says the universe is only 13.7 billion years old, just to give you an idea.

So, at this point the computer science fails us. We require a better algorithm.

Fortunately, we can come up with one easily. Noting that $2048 = 2^{11}$, it is quite clear that, instead of calculating

$$a^{2^{2048}}$$

we can instead calculate

$$a^{2^{2^{11}}}$$

which at least involves smaller numbers.

As written, the right associativity of exponentiation means we can't really start by calcuating 2^{11} and take 2 to that power, and so on; that leads us right back to where we started.

What we can do, though, is deal with 2^{11} as a product of eleven 2s. We do have a rule for powers where the exponent is a product; that

$$2^{2^{11}}$$

then becomes

$$((((((((((2^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2$$

and we can now see a very clear way to calculate this number by computer; instead of repeatedly multiplying by a, we repeatedly square. This cuts our work down from 2^{11} operations (multiplications by a) to just 11 (squarings).

We now can make this algorithm more general; after all, we don't just want to calculate $2^{2^{11}}$, but

$$a^{2^{2^{11}}}$$

which is more abstract. Suppose we need to calculate x^k using this quicker idea; as we have just shown, if k is even, we can calculate $x^{k/2}$ and square. If k is odd, we can calculate $x \times x^{k-1}$, and, as $k-1$ is even, we can use squaring again. Note, too, that we are promised an end to this computation, as k is a positive integer and repeatedly subtracting 1 and halving must eventually give 1, and $x^1 = x$ by definition.

We now need language to describe this situation. The simplest way to deal with algorithms in this simple situation (namely, where there is no recursion) is to consider the algorithm to be simply a finite sequence of additions, subtractions, multiplications, and divisions of integers. (Note: we consider "integer division" in this sense to involve ignoring any remainder.)

Now, the real world does intrude here a bit. In the building of computers, circuits that add are very simple to build; in the jargon of electrical engineering, a one-bit full adder can be built with only two common gates. Subtraction is also easily accomplished with recourse to two's complement notation for negative numbers, which lets the same circuit add and subtract. The relevant circuit is also nearly trivial to build.

What hurts is multiplication and division. These operations take significant time, and so we perform rudimentary complexity analysis by counting how many multiplications and divisions occur in our algorithm. Extracting a residue is in this same bracket, being somewhat complementary to integer division.

Thus, consider applying an algorithm beginning with a given input x. In the case of our more naïve first algorithm to calculate $x^k \bmod N$, there will be $k-1$ multiplications and $k-1$ reductions modulo N, for a total of $2k-2$.

Now, the second algorithm, based on squaring, is more difficult; sometimes we multiply by x, other times we square. One small mercy we can find is that the binary expansion of k tells us the pattern of squaring and

multiplying, and we can consider the worst, best, and average cases. If k is of the form $2^j - 1$, then the binary expansion of k will be all 1s, and we will have to perform j multiplications, j squarings, and j reductions modulo N. If we are more lucky and find that $k = 2^j$ for some j, then we never have to multiply by x at all; we just square j times and reduce by N j times. On average, then, we might expect that we have to multiply by x $\frac{j}{2}$ times. All told, this algorithm would then take between $2j$ and $3j$ slow operations; probably right around $2.5j$ if we had to make an exact guess.

The key to this — where we see the efficiency — is that $j = \lfloor \log_2 k \rfloor + 1$. More easily, $j < \log_2 k + 1$, which we may as well call $(1 + \epsilon) \log_2 k$. The squaring based algorithm then takes $2.5(1 + \epsilon) \log_2 k$ operations.

Now, this is the rather odd bit that is responsible for a great deal of confusion between mathematicians and computer scientists. To a computer scientist, the universe is made up of strings of *symbols*, not numbers. A mathematician, particularly dealing with number theory, will normally see numbers, and so the two disciplines have naturally disagreeing ideas of size. In mathematics and number theory, the size of a number N is simply N, maybe $|N|$ if we're allowing negatives. To a computer scientist, N must be represented by a string of symbols; maybe the digits zero through nine, maybe just the digits zero and one. Either way, the size of N in this light is the number of symbols required, which we usually estimate as $\log_2 N$.

We therefore express that estimate of $2k - 2$ as $22^{\log_2 k} - 2$. This is exponential in $\log_2 k$, so we describe the naïve algorithm for modular exponentiation as having *exponential running time*. Our new algorithm, which takes approximately $2.5 \log_2 k$ operations, is said to have *linear running time*. [CLRS01]

The reason this is sticky is that mathematicians see the expression $2k - 2$ and, for good reason, think that this should represent linear time; $2.5 \log_2 k$ should, similarly, be exponential.

It is, of course, worth making a general definition here, but making it completely formal is too tall a task for our purposes; the precise definition of "algorithm" can get thorny fast, unless we're willing to start talking about things like Turing machines or the lambda calclus. This definition, then, must be taken with some salt.

Simple Estimates on Running Time of Algorithms 1. Let A be an iterative computational algorithm; that is, a sequence of basic operations on natural numbers that may or may not branch or loop.

Suppose that there exists a function f with the property that, when A is applied to the number n, $f(\log_2 n)$ is greater than the number of multiplications, divisions, and residue extractions that occur in the computation of $A(n)$.

Conventions

When f is linear, we say A has *linear running time*. When f is polynomial, we say A has *polynomial running time*, or, more succinctly, *A is in P* or A is *feasible*. When f is exponential, we say A has *exponential running time*.

Now, the real reason we care about running time is that, historically speaking, the "sweet spot" for computer programs is at polynomial running time. We think of such algorithms as scaling well, for instance; even if the input is long, the program will at least stay in the bracket of minutes and hours instead of centuries and milennia. This is, in part, why we use such a loaded word as "feasible".

At this point, then, we have a feasible computation for a^N in $\mathbb{Z}_N$. This is a point in favor for the Amended Fermat Test; it might not be perfectly accurate, but it's definitely easy. The next step is to see what else this idea buys us.

5 The Miller-Rabin Test

In some sense, number theory begins with modular squaring; one of the first big nontrivial theorems of the discipline is Gauss' Quadratic Reciprocity Theorem. One would hope that we would be able to use tools of this level to help us prove numbers prime or composite.

As before, we have our candidate prime N, and we can look at the calculation of a^N mod N in a new light.

Firstly, N will be odd, so we should take our original test, that $a^N = a$ in $\mathbb{Z}_N$, and divide out by a so that we have $N-1$ for the exponent; this will be even, thus at least letting us start the algorithm by squaring. This gives an alternate, slightly less valid form, but for our purposes, much improved. We now wish to prove that $a^{N-1} = 1$ in $\mathbb{Z}_N$, and $N-1$ is even.

In fact, let's take all we can get. Let $N - 1 = 2^r s$, where s is odd. We can then compute a^s, and square r times to get a^{N-1}.

Right now this is still just the Amended Fermat Test, though; there's a lot more here, if we're willing to look.

We can consider the starting point to be a^s, and then we square t times. We therefore have t values, the last of which may or may not be 1; if it is, then we at least have a-pseudoprimality.

However, what happens if we find an anomalous square root of 1 in this process? 1 and -1 are fine, but what if we should find that $a^{2^k s} \neq \pm 1$ and $a^{2^{k+1} s} = 1$? Letting $z = a^{2^k s}$, we get that $z \neq \pm 1$, but $z^2 = 1$. Given that all this is true in $\mathbb{Z}_N$, we have $N \not| z - 1$, $N \not| z + 1$, but $N | z^2 - 1$. Given that $z^2 - 1 = (z + 1)(z - 1)$, this forces N to be composite; N divides a product without dividing either factor, which is the ring-theoretic (and more general) definition of primality. For instance, note that $5^2 = 1 \pmod{24}$, and this tells us that $5^2 - 1 = (5 - 1)(5 + 1)$ divides 24, giving us two divisors, 6 and 4.

This is one of the two steps up we take in the Miller-Rabin algorithm. Instead of simply calculating a^N, we now calculate a^s, and square $t-1$ times. This will prove N to be composite if this process either does not end in 1 or -1, or if a 1 occurs midsequence (if a^s is 1, that's fine) without a preceding -1.

The other matter is what we do with a in this test. As we earlier showed, there exist things like pseudoprimes.

It is somewhat regrettable, but the easiest way around this — and the other step to the Miller-Rabin algorithm — is to randomize the test. This is unfortunate, because it turns our test from a deterministic one into what is termed a Monte Carlo algorithm; with some nonzero probability, it gives the wrong answer.

There are mitigating factors, though; the Miller-Rabin algorithm is reliable insofar as proving compositeness. We never can prove a number prime with this test, though the algorithm gives us a solid proof when it states that a number is composite.

Practically speaking, the way we use the Miller-Rabin test is simply to repeat it several times, looking for some value of a that will prove N composite; we call such numbers *witnesses*.

At this point, then, we can give the complete algorithm. [Rab80]

The Miller-Rabin Algorithm 1.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

```
/*A simple implementation of fast modular exponentiation;
 given x, k, and N, this function will return x^k mod N. Note
 that this function, speaking properly, should be implemented
 via iteration, not recursion. However, a good compiler should
 make such a change as part of optimization, so this version
 is provided in the interest of simplicity.*/
unsigned int modexp (unsigned int x,
     unsigned int k,
     unsigned int N )
{
  if (1 == k) /*terminating condition*/
    return x % N;

  if (1 == k % 2) /*k is odd: extract a factor of x*/
    return x * modexp(x , k - 1 , N) % N;
  else /*k is even: square*/
    return (unsigned int)pow(modexp(x , k/2 , N) , 2) % N;
}

int MillerRabin ( unsigned int N,  /*The number to test for
                                     primality*/
  unsigned int k ) /*The number of random tests to
                                     run*/
{
  int a = 0; /*The random number*/
  int r = 0, s = N - 1; /*r and s are used to decompose N;
                          N = s * 2^r + 1*/
  int x1, x2; /*These hold the values we get from squaring.*/
  int i, j; /*loop indices*/

  srand(time(NULL)); /*We need to seed the random number
                       generator. At least that's good form
                       here.*/

  while (0 == s % 2) /*Extract the values of r and s from N.*/
    {
      r++;
```

```
      s /= 2;
    }

  for (i = 0; i < k; ++i)
    {
a = rand() % N; /*Get a (pseudo)random number.*/
while (0 == a) /*We also need to be certain that a is
                        nonzero.*/
  a = rand() % N;

x1 = modexp(a,s,N);

for (j = 0; j < r; ++j)
  {
    x2 = (unsigned int)pow(x1 , 2) % N;
    if (( 1 == x2 ) && (( 1 != x1) && ( N - 1 != x1 )))
      return 0;
    x1 = x2;
  }
if (1 != x2 )
  return 0;
    }
  return 1;
}
```

5.1 Analysis

Now, there are some questions we should ask about such a lovely piece of code.

Perhaps the most obvious is "Is the Miller-Rabin test any more accurate than the Amended Fermat Test?". On one level, the answer is an exceptionally easy "yes"; it should be obvious that randomizing a should enable the Miller-Rabin test to, with enough iterations, detect any composite number that isn't absolutely pseudoprime.

In fact, for that matter, how does the Miller-Rabin test perform against absolute pseudoprimes?

```
$./MillerRabin 561 12
```

```
561 is composite
$
```

Not the result we might have expected. After all, Miller-Rabin has its roots in the Amended Fermat Test, which is fooled by absolute pseudoprimes. In fact, 7 serves as a fine witness to the compositeness of 561; $7^{280} \bmod 561 = 67$, but, as 561 is an absolute pseudoprime, $7^{560} \bmod 561 = 1$, so we get 67 as a nontrivial square root.

However, note that our condition on nontrivial square roots is new. This condition is powerful enough to detect even the compositeness of absolute pseudoprimes, which, on a moment's thought, is not too surprising; if we recall the mechanism by which nontrivial square roots prove primality, that has very little to do with the content of Korselt's Criterion.

More difficult is the question of error. The Miller-Rabin algorithm is, after all, a Monte Carlo algorithm; it can be incorrect. We'd like to characterize those failures as best we can, though, because we're dealing in probabilities, it's not going to be quite as clear as our results on pseudoprimality. What we find is that, given any N — even an absolute psuedoprime — at least $\frac{3}{4}(N-1)$ of the numbers from 1 to N will be witnesses. [Rab80]

For practical purposes, this is more than enough; we can effectively generate random primes by applying the Miller-Rabin test to the output of a random number generator. While this may not be the quickest process ever seen, it isn't too bad, and, furthermore, we know we're free of any concerns about absolute pseudoprimes. In this context, it's okay to set k relatively low, say, as low as 6; the probability of error is about $4^{-6} = \frac{1}{4096}$; just under 0.025%. In fact, the probability of a random number failing is probably much less.

However, one practical concern is that in many modern cryptographic protocols, the participants, Alice and Bob, are called upon to exchange prime numbers, and an attacker, Eve, can subvert the protocol by posing as Alice and submitting a composite number in place of Alice's prime.

In such a situation, that bound of 4^{-k} is strict, as Eve can choose her "random" number so that the probability of failure is maximal, nearly 4^{-k}. In an application like this, in order to prevent compromise, several dozen rounds of the Miller-Rabin algorithm are more appropriate.

6 The AKS Algorithm

So, right now, the state-of-the-art in primality testing is a probabilistic algorithm that requires a measure of subtlety to apply. Granted, this is sufficient for practical purposes; the Miller-Rabin algorithm is certainly fast enough for modern computers and cryptosystems.

However, we'd really like some icing for the cake, so to speak; we'd like a deterministic algorithm that has all of the above properties, too.

Such an algorithm was, in fact, recently discovered by three Indian mathematicians. [AKS04]

6.1 Core Mathematics

The problem with primality tests based on Fermat's Little Theorem, mathematically speaking, is that Fermat's Little Theorem is not a bidirectional implication; we have only a necessary condition for primality, not a sufficient one.

What we need, therefore, is a condition equivalent to primality that is more amenable to calculation (as opposed to searching).

The simplest way to obtain such a condition is to note that part of the problem may be that our ring $\mathbb{Z}_N$ is too small. We may therefore move up to the polynomial ring $\mathbb{Z}_N[x]$ and consider what primality of N means in this ring.

Primality in Polynomial Rings 1. *The following are equivalent:*

1. *N is prime.*
2. *In the polynomial ring $\mathbb{Z}_N[x]$, $(x+a)^N = x^N + a$ for all $a \in \mathbb{Z}_N$.*
3. *In $\mathbb{Z}_N$, $\binom{N}{k} = 0$ for $1 \leq k < N$ and $a^N = a$.*

We first prove that the first condition implies the second and third. In general, we have

$$(x+a)^N = \sum_{k=0}^{N} \binom{N}{k} x^{N-k} a^k$$

which is a simple application of the Binomial Theorem.

We can also very easily extract the terms corresponding to $k = 0$ and $k = 1$, which give x^N and a^N, and the latter, by Fermat's Little Theorem, is equal to a modulo N. We therefore have at this point that

$$(x + a)^N = x^N + a + \sum_{k=1}^{N-1} \binom{N}{k} x^{N-k} a^k$$

and we can now prove that $N|\binom{N}{k}$ for all $1 \leq k < N$.

By definition, $\binom{N}{k} = \frac{N!}{k!(N-k)!}$. By hypothesis, N is prime, which means that, as $1 < k < N$, neither k nor $N - k$ can be divisors of N. In fact, $k!$ and $(N-k)!$, being products of numbers between 2 and $N-1$, must also be relatively prime to N. This means that, regardless of k, $\frac{N}{k!(N-k)!}$ will always have a factor of N in the numerator, and therefore $\binom{N}{k}$ will always be a multiple of N when N is prime.

This allows us to conclude that, in $\mathbb{Z}_N$, $\binom{N}{k} = 0$ for all nonzero k, and therefore that

$$\sum_{k=1}^{N-1} \binom{N}{k} x^{N-k} a^k = 0$$

and, ultimately, $(x+a)^N = x^N + a$. Therefore, the first condition implies the second.

We also need to prove that the second condition implies the first, which is most easily done by contraposition; we assume N is composite, and use a proper nontrivial divisor c of N to find a nonzero term in the polynomial $(x + a)^N$.

Suppose $c^j || N$; that is, $c^j | N$ but $c^{j+1} \not| N$. We can quickly see that $c^j || \frac{N!}{(N-c)!}$, as $\frac{N!}{(N-c)!}$ is a product of exactly c numbers, so only one of them — N — can be a multiple of c. It follows from this that $c^{j-1} || \binom{N}{c}$, and thus that $c^j \not| \binom{N}{c}$. This forces our hand; it must be that $N \not| \binom{N}{c}$, as $N | \binom{N}{c}$ would imply that c^j did divide $\binom{N}{c}$.

Considering the case $a = 1$, we then find that the coefficient of degree c in $(x + 1)^N$ will be nonzero, therefore proving that the first and second conditions are equivalent.

Proving equivalence of the second and third conditions is quite simple; in fact, we've basically done it already, simply by applying the binomial theorem and considering the case $a = 1$. $\square$

This does our job quite nicely; we have two conditions now, one algebraic and the other more combinatorial in character, that are equivalent to primality.

There is, however, a very large problem with both of these new conditions.

For the algebraic condition, we have to handle a polynomial of degree N. Computationally, this gets represented by a point in $\mathbb{Z}^N$ (in computer parlance, an *array*), which ends up being an awful lot of data when N gets large; computing arrays of 2^{2096} elements can take a very long time.

The same logic applies to the combinatorial formulation; we have to calculate $\binom{N}{k}$ for $N-1$ values of k; that's far too much math to be able to do in any reasonable timeframe. (However, the nice thing about the combinatorial formulation, it should be said, is that it very nicely highlights the fact that this theorem is an extended version of Fermat's Little Theorem.)

Now, this gives us one half of the big solution; we have a computational, but inefficient solution. Now we have to come up with the speed.

6.2 Finding The Right Ring

See, part of the problem is that we've gone too far. Algorithmically, $\mathbb{Z}_N$ is a fantastic ring, by which we mean finite; it's possible to search in a very clean fashion. Going up to $\mathbb{Z}_N[x]$ costs us that finiteness; now we might run into lots of really big polynomials, which means long arrays, and everything can go right back to the fire. Our running time needs to be polynomial in $L = \log_2 N$ in order to keep this under control.

What we need is a more general idea than polynomial rings. One option is to think of $R[x]$ as the smallest ring containing R and x; we refer to this process as *adjunction*, and refer to $R[x]$ as the ring formed by *adjoining* the indeterminate x to the ring R.

The idea behind this is that we can adjoin all sorts of things to R besides indeterminates. A popular choice — the one we make here — is to adjoin roots of unity. We can form $\mathbb{Z}_N[\zeta_r]$, where ζ_r is a (primitive) rth root of unity, and r and N are coprime. This ring has N^r elements, which, while quite a lot, is at least finite; we may be able to skate by if we choose r just right. (Note: if the notation $\mathbb{Z}_N[\zeta_r]$ makes you skittish, it is equally correct to think of it as $\mathbb{Z}_N[x]/q(x)$, where q is an irreducible nonlinear polynomial dividing $x^r - 1$; the Fundamental Isomorphism Theorem very easily proves these two identical, as long as r and N are coprime.)

This is the key to the entire algorithm; not to do the arithmetic in $\mathbb{Z}_N$, where we have to search and use Fermat's Little Theorem and suffer weakness, nor in $\mathbb{Z}_N[x]$ where we have all the power we need and too many elements, but in $\mathbb{Z}_N[\zeta_r]$, where we might be able to pull a Goldilocks and make things just right. The tricky part will be in choosing the right value of r; in fact, right now, most of the work on improving the algorithm is in finding the right definition of r.

In some sense, we need to make sure that r is big enough; say, for instance, that the cyclic subgroup of $\mathbb{Z}_N$ generated by r is large.

Choosing R 1. *Once $N > 4$, there exists $r < L^5$ such that $N^t \neq 1 \pmod{r}$ for all $t \leq L^2$.*

Consider all nondivisors of the product

$$\prod_{t=1}^{L^2} \left(N^t - 1\right)$$

and let r be the smallest such. It is clear that r does not divide any factor $N^t - 1$, which makes it clear that $N^t \neq 1$ for all $t \leq L^2$.

It remains to prove that $r < L^5$.

Note that

$$\prod_{t=1}^{L^2} \left(N^t - 1\right) < \prod_{t=1}^{L^2} N^t$$

and we can express this second, much simpler product, as a power of N; it is $N^{L^2(L^2+1)/2}$.

Now, note that once $N > 4$, we have $L > 2$, and now $L^2 < L^4$, so $L^4 + L^2 < 2L^4$ and therefore $L^2(L^2+1)/2 < L^4$.

This then indicates that $N^{L^2(L^2+1)/2} < N^{L^4}$, and, as $L = \log_2 N$, $N^{L^4} = 2^{L^5}$.

At this point we need a lemma, the proof of which is a particularly clever bit of analytic number theory. [Nai82]

A Cheybyshev-Style Result on Products of Primes 1. *The product of all primes less than or equal to α is at least 2^α.*

Given $m < n$, consider the integral

$$I_{m,n} = \int_0^1 x^{m-1}(1-x)^{n-m} \, \mathrm{d}x$$

which, on application of the Binomial Theorem, is

$$\sum_{j=0}^{n-m} (-1)^j \binom{n-m}{j} \frac{1}{m+j}$$

and, letting P_α be the product of all primes less than or equal to α, the distributive property makes it quite clear that $I_{m,n}P_n$ is a natural number, as the fraction in the sum goes from $\frac{1}{m}$ to $\frac{1}{n}$.

However, note too that calcluating $I_{m,n}$ via integration by parts with $u = x^{m-1}$ and $\mathrm{d}v = (1-x)^{n-m}\,\mathrm{d}x$ gives the recurrence relation $I_{m,n} = \frac{m-1}{n-m+1} I_{m-1,n}$. Applying this along with the base case $I_{1,n} = \frac{1}{n}$, indicate that $I_{m,n} = \frac{1}{m\binom{n}{m}}$.

In order for $I_{m,n}P_n$ to be an integer, then, it must be that $m\binom{n}{m}$ divides P_n, regardless of m. We can conclude from this that $n\binom{2n}{n} | P_{2n}$, and, as $(2n+1)\binom{2n}{n} = (n+1)\binom{2n+1}{n+1}$, we also get that $(2n+1)\binom{2n}{n}$ must divide P_{2n+1}.

So, as $P_{2n} | P_{2n+1}$, it follows that both $n\binom{2n}{n}$ and $(2n+1)\binom{2n}{n}$ divide P_{2n+1}. As n and $2n+1$ are coprime, we must have $n(2n+1)\binom{2n}{n}$ dividing P_{2n+1}.

This then means that P_{2n+1} must be at least $n(2n+1)\binom{2n}{n}$, and, if we're clever enough to see that $\binom{2n}{n}$ is the largest of the $2n+1$ coefficients in $(1+1)^{2n}$, we can note that $n(2n+1)\binom{2n}{n} > n4^n$.

At this point we have that $P_{2n+1} > n4^n$. When $n > 2$, it is quite clear from this that $P_{2n+1} > 2^{2n+1}$. Similarly, when $n > 4$, we know that $P_{2n+2} > P_{2n+1} > 2^{2n+2}$. Thus, for all α, both even and odd, we have $P_\alpha > 2^\alpha$, as desired. $\square$

At this point, we know that our product

$$\prod_{t=1}^{L^2} \left(n^t - 1\right)$$

is less than 2^{L^5}. Our lemma then tells us that a nondivisor must exist among the numbers less than L^5, and therefore the existence of our necessary $r < L^5$ is guaranteed. $\square$

We can now use r to perform a first check on N; all we do is compute (t, N) where $2 \le t \le r$. Should this ever be greater than 1, we have a divisor and N is composite.

There is also the question of what to do when $r \ge N$; granted, as $r < L^5$, this is a rare condition that only can occur for smaller N, where $N <$

$(\log_2 N)^5$ — past five million or so, this no longer is a concern. Furthermore, by dealing with this small case directly, it actually is faster than applying the AKS algorithm proper.

At this point, then, we know that we have $r < L^5$ coprime to N, and the useful property of r is that for all $t < L^2$, $N^t \neq 1 \pmod{r}$.

Now, one of the things this means is that there is a prime number p that divides N for which $p^t \neq 1$ for all $t < L^2$, as can be seen by expressing N as a (possibly trivial) product of primes. It is also true that $p > r$; otherwise, we would detect the divisibility when we checked (p, N) earlier. Put simply, we now have two avenues on which we are starting to pin down a prime divisor of N.

As regards p, note that we cannot actually get our hands on it at this point. However, we can identify $\mathbb{Z}_{N/p}$ as an ideal of $\mathbb{Z}_N$, and thus $\mathbb{Z}_p[\zeta_r]$ as an ideal of $\mathbb{Z}_N[\zeta_r]$. Again, we can't actually compute in this ring, as we have no algorithm yet to find p, but we can talk about results in this ring, as the fact that p is prime makes things go much smoother.

6.3 Introspective Numbers

For one, note that, while we know that $(x + a)^N = x^N + a$ in $\mathbb{Z}_N[\zeta_r]$ by hypothesis, we also know that $(x + a)^p = x^p + a$ in $\mathbb{Z}_p[\zeta_r]$. In fact, because p divides N, we know that $(x + a)^N = x^N + a$ must also be true in $\mathbb{Z}_p[\zeta_r]$. From this we can deduce that, where $c = \frac{N}{p}$, $(x + a)^c = x^c + a$ in $\mathbb{Z}_p[\zeta_r]$.

Abstracting away from this a bit, we have a polynomial $P \in \mathbb{Z}_p[\zeta_r]$ and several natural numbers k (N, p, and c) for which $P(x^k) = (P(x))^k$. In such a situation as this (and, it should be noted, this extremely limited scope), we will call such numbers *P-introspective*.

What makes this property important is the following theorem

Properties of Introspective Polynomials 1. *For any fixed polynomials P and Q in $\mathbb{Z}_p[\zeta_r]$, the following hold:*

- *Products of P introspective numbers are P-introspective.*
- *Numbers that are both P- and Q-introspective are PQ-introspective.*

Let k and j be P-introspective; we have $P(x^k) = (P(x))^k$ and $P(x^j) = (P(x))^j$. Taking powers on this last equation gives $(P(x^j))^k = (P(x))^{jk}$; thus, if we can prove $P(x^{jk}) = ((P(x^j))^k$, we will be done.

Treating $\mathbb{Z}_N[\zeta_r]$ as $\mathbb{Z}_N$ modulo the ideal generated by the irreducible polynomial q dividing $x^r - 1$ and replacing x with x^k in the definition of k's P-introspectivity, we find that we do have $P(x^{jk}) = (P(x^j))^k$, but in the ring $\mathbb{Z}_N[\zeta_{kr}]$.

The final step is then realizing that $\mathbb{Z}_N[\zeta_r]$ is a subring of $\mathbb{Z}_N[\zeta_{kr}]$, so we at last have P-introspectivity of jk.

Finally, let k be both P- and Q-introspective. Straightforward calculation gives us $P(x^k)Q(x^k) = (P(x))^k(Q(x))^k = (P(x)Q(x))^k$, so k is also PQ-introspective. $\square$

The reason this is important is that we have p- and c-introspectivity for every monic linear polynomial in $\mathbb{Z}_N[\zeta_r]$; thanks to this theorem, we now have a lot more. We now know that any polynomial in $\mathbb{Z}_N[\zeta_r]$ that is a product of monic linear polynomials is $c^i p^j$-introspective for any natural numbers i and j.

Let's look a little more closely at these numbers $c^i p^j$. As $c = \frac{N}{p}$, and N is coprime to r, note that $c^i p^j$ is therefore also coprime to r. We can therefore talk about the subgroup H of $\mathbb{Z}_r^\times$ generated by c and p.

Now, recall that we chose r particularly so that we had $N^t \neq 1$ for all $t < L^2$, and, in fact, it is equally true that $p^t \neq 1$ for all $t < L^2$, as previously discussed. The punchline to this joke is that this now forces it to be the case that, as p^t is always in H, that the order of H is at least L^2. Letting $h = |H|$, we have $h > L^2$.

The other group G is that generated by all the monic linear polynomials in $\mathbb{Z}_N[\zeta_r]$. We need to know that all the polynomials of degree less than h are distinct, and this follows very simply by noting that $h < r$,

At this point we also introduce the final bound we need, $\ell = \sqrt{\varphi(r)}L$; this will be the values of a for which we confirm $(x + a)^N = x^N + a$. Noting that $\ell < N$, we can deduce that the family of $\ell + 1$ nonzero monic linear polynomials $x + a$, where $0 \leq a \leq \ell$, are all distinct and can be used to generate G. If we restrict ourselves to polynomials of degree less than h, then we have

$$\sum_{i=0}^{h} \binom{\ell + i}{i}$$

and a little combinatorics and Pascal's Triangle will show that this is $\binom{h+\ell}{h-1}$. At last, then, we know that the order of G is at least $\binom{h+\ell}{h-1}$.

On the other hand, if N is not a prime power, we can bound $|G|$ from the other direction. Consider the subset of H made up of numbers of the form

$c^i p^j$ where i and j are both less than or equal to $\sqrt{h}$. If N is not a prime power, then this set must contain at least $(\sqrt{h}+1)^2$ elements. Noting that this is strictly greater than t, this set must contain two elements that are congruent modulo r. Calling these k and k', with $k > k'$, we find that, in $\mathbb{Z}_N[\zeta_r]$, $x^k = x^{k'}$.

This immediately implies that, for any polynomial $P \in G$, $(P(x))^k = (P(x))^{k'}$. In other words, any polynomial in G divides the polynomial $x^k - x^{k'}$. In other words, the distinct roots of $x^k - x^{k'}$, which number at least k, and $k \leq (cp)^{\sqrt{h}} = N^{\sqrt{h}}$. Therefore, when N is not a prime power, $|G| \leq N^{\sqrt{h}}$.

We can now complete the proof of correctness of the algorithm. We have $\binom{h+\ell}{h-1} \leq |G| \leq N^{\sqrt{h}}$. As $h > \sqrt{h}L$, we get $\binom{h+\ell}{h-1} \geq \binom{\ell+1+\sqrt{h}L}{\sqrt{h}L}$. As $\ell > \sqrt{h}L$, too, we actually get $|G| \geq \binom{2\sqrt{h}L+1}{\sqrt{h}L}$. This is greater than $2^{\sqrt{h}L+1}$, once we take the floor of the involved numbers, we'll get at least $\sqrt{h}L + 1$ factors of 2. And, finally, as $L = \log_2 N$, we get $|G| > N^{\sqrt{h}}$.

This contradiction thus forces it to be the case that N is a prime power, which we can easily check for.

For clarity, then, the entire algorithm goes as follows.

Given: N, run simple checks to see if N is a prime power. If not, take $r < (\log_2 N)^5$ so that $N^t \neq 1 \bmod r$ for all $t < (\log_2 N)^2$. Compute (t, N) for all $t < r$, and if this is ever greater than 1, N is composite. (For small N, it may be the case that $N \leq r$; if this happens, N is prime.) At last, verify that $(x+a)^N = x^N + a$ in $\mathbb{Z}_N[\zeta_r]$ for $1 \leq a \leq \sqrt{\varphi r}\log_2 N$; this is true if and only if N is prime.

We therefore have proven the soundness of the AKS algorithm. Note, though, that it is simply too slow. We have to work with $\sqrt{\varphi(r)}L$ polynomials, each of which have degree r and, on average, coefficients of size L. Given that $r < L^5$, Each computation will require L multiplications, so, at last, we have $r\sqrt{\varphi r}L^3 < L^{10.5}$ as our bound on the running time. This is far too slow to completely supplant Miller-Rabin.

However, the AKS algorithm is as yet being improved, and contains a great deal of insight that may yet yield improvements in the field of numerical algorithms.

References

[AKS04] MR2123939 (2006a:11170) Agrawal, Manindra ; Kayal, Neeraj ; Saxena, Nitin . PRIMES is in P. Ann. of Math. (2) 160 (2004), no. 2, 781–793.

[CLRS01] MR1848805 (2002e:68001) Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford . Introduction to algorithms. Second edition. MIT Press, Cambridge, MA; McGraw-Hill Book Co., Boston, MA, 2001. xxii+1180 pp. ISBN: 0-262-03293-7

[Nai82] MR0643279 (83f:10043) Nair, M. On Chebyshev-type inequalities for primes. Amer. Math. Monthly 89 (1982), no. 2, 126–129.

[Rab80] MR0566880 (81f:10003) Rabin, Michael O. Probabilisic algorithm for testing primality. J. Number Theory 12 (1980), no. 1, 128–138.

[Car10] Carmichael, R.D. Note on a new number theory function. Bull. Amer. Math. Soc. 16 (1910), 232–238