Verification of the Miller-Rabin Probabilistic Primality Test

Joe Hurd*

Computer Laboratory
University of Cambridge
joe.hurd@cl.cam.ac.uk

Abstract. We use our HOL formalization of probability theory to specify and verify a version of the Miller-Rabin probabilistic primality test. The version of the test that we implement is guaranteed to terminate and requires only a source of random bits, but satisfies the same probabilistic specification as the abstract version presented in algorithm textbooks. In the course of the verification we formalize a large body of computational number theory, which is used to evaluate our predicate subtype prover. The verified version of the algorithm is then manually extracted to Standard ML, and applied to some examples.

1 Introduction

In this paper we define in HOL a probabilistic function miller_rabin and prove the following two properties of it:

$$\vdash \forall n, t, s.\ \mathsf{prime}\ n\ \Rightarrow\ \mathsf{fst}\ (\mathsf{miller\_rabin}\ n\ t\ s) = \top \tag{1}$$

$$\vdash \forall n, t.\ \neg(\mathsf{prime}\ n)\ \Rightarrow\ 1 - 2^{-t} \leq \mathbb{P}\,\{s : \mathsf{fst}\ (\mathsf{miller\_rabin}\ n\ t\ s) = \bot\} \tag{2}$$

The miller_rabin function takes two natural number parameters n and t (in addition to a random sequence s): if n is prime then it is guaranteed to return $\top$; if n is composite then it will return $\bot$ with probability at least $1 - 2^{-t}$. Thus for a given value of n if miller_rabin $n\ t\ s$ returns $\bot$ then n is definitely composite, but if it returns $\top$ then all we know is that n is probably prime.[1] However, setting $t = 50$ we see that the probability of the algorithm returning $\top$ for an n that is actually composite is $\leq 2^{-50} < 10^{-15}$.

This algorithm is used to test large numbers for (probable) primality in computer algebra systems such as Mathematica, and it is also relevant to public key cryptography software (the RSA algorithm requires a modulus of the form $n = pq$ where p and q are primes).[2]

* Supported by an EPSRC studentship

[1] Quantifying that 'probably' is a hard problem: the probability that n is prime given that miller_rabin $n\ t\ s$ returned $\top$ depends on the set S from which n was chosen and the distribution of primes in S.

[2] Surprisingly, the popular email encryption program PGP (and the Gnu version GPG) use the Fermat test to check numbers for primality, although the Miller-Rabin test is stronger and involves no extra computation.

We used the theorem-prover hol98 to perform the verification, and the novelty lies in the fact that this is an algorithm with a probabilistic specification used in commercial software. We build on our earlier work [7] and show that the formal probability framework we layed out is up to the challenge. In addition, we present a version of the verified algorithm that we have manually extracted to ML. This is not quite identical to the version found in algorithm textbooks, since they generally assume a generator that can produce uniformly distributed random numbers in the range $\{0, \ldots, n-1\}$, while our version is guaranteed to terminate and assumes only a generator of random bits.[3] The difference is that while 'genuine' random bits are provided by most operating systems (e.g., in Linux from `/dev/random`), the generation of uniformly random numbers from these random bits is slightly delicate.[4]

A significant amount of number theory is necessary to verify the algorithm, and Sect. 2 shows how the classical results fit together in the verification. This formalization actually constituted the bulk of the effort, and provided the testing ground for our predicate subtype prover. Section 3 describes the somewhat easier task of interfacing the number theory with the probability theory to produce the result, and then in Sect. 4 we examine the algorithm extracted to ML. Finally in Sects. 5–7 we conclude, consider various extensions and look at related work.

1.1 Notation

We use sans serif font to notate HOL constants, such as the function fst that picks the first component of a pair, the natural number predicate prime, the greatest common divisor function gcd, and the group predicate cyclic. For standard mathematical functions we use mathematical font: examples are addition $(a + b)$, the remainder function $(a \bmod b)$, function composition $(g \circ f)$ and Euler's totient function $(\phi(n))$. We rely on context to disambiguate $|S|$ to mean the cardinality of the set S, $|g|$ to mean the order of the group element g, and $a \mid b \mid c$ to mean that both a divides b and b divides c. When doing informal mathematics, we follow the convenient custom of confusing the group G with its carrier set; in HOL theorems we explicitly write set G for the carrier set (and $*_G$ for the operation).

2 Computational Number Theory

2.1 Definitions

Our definition of the Miller-Rabin algorithm is a functional version of the one presented in Cormen, Leiserson and Rivest [4]. To prepare, we define functions to

[3] In this paper, a *"generator of random bits"* means an infinite sequence of independent identically distributed (IID) Bernoulli($\frac{1}{2}$) random variables.

[4] Concretely, it is impossible for a terminating algorithm to construct random numbers uniformly distributed in the range $\{0, \ldots, n-1\}$ from random bits unless n is a power of 2 [7].

factor powers of 2 and perform modular exponentiation. Here are the correctness theorems for these functions:

$$\vdash \forall n, r, s.\ 0 < n \ \Rightarrow\ (\mathsf{factor\_twos}\ n = (r, s) \iff \mathsf{odd}\ s \ \wedge\ 2^r s = n) \quad (3)$$

$$\vdash \forall n, a, b.\ 1 < n \ \Rightarrow\ \mathsf{modexp}\ n\ a\ b = (a^b \bmod n) \quad (4)$$

Next we define a function $\mathsf{witness}\ a\ n$ that is completely deterministic, simply returning $\top$ if the base a is a 'witness' to the compositeness of n and $\bot$ otherwise. We assume that a and n satisfy $0 < a < n$. The witness function uses a helper function witness_tail which is defined using pattern-matching:

$$\begin{aligned}
&\vdash (\forall n, a.\ \mathsf{witness_tail}\ n\ a\ 0 = a \neq 1) \ \wedge \\
&\quad (\forall n, a, r. \\
&\qquad \mathsf{witness_tail}\ n\ a\ (\mathsf{suc}\ r) = \\
&\qquad \mathsf{let}\ a' \leftarrow (a^2 \bmod n) \\
&\qquad \mathsf{in\ if}\ a' = 1\ \mathsf{then}\ a \neq 1 \ \wedge\ a \neq n - 1 \\
&\qquad\quad \mathsf{else\ witness_tail}\ n\ a'\ r) \qquad (5) \\
&\vdash \forall n, a. \\
&\qquad \mathsf{witness}\ n\ a = \\
&\qquad \mathsf{let}\ (r, s) \leftarrow \mathsf{factor_twos}\ (n - 1) \\
&\qquad \mathsf{in\ witness_tail}\ n\ (\mathsf{modexp}\ n\ a\ s)\ r \qquad (6)
\end{aligned}$$

The witness function calls factor_twos to find r, s such that s is odd and $2^r s = n - 1$, then uses modexp and witness_tail to calculate the sequence

$$(a^{2^0 s} \bmod n, a^{2^1 s} \bmod n, \ldots, a^{2^r s} \bmod n)$$

This sequence provides two primality tests for n:

1. $a^{2^r s} \bmod n = 1$.
2. If $a^{2^j s} \bmod n = 1$ for some $0 < j \leq r$, then either $a^{2^{j-1} s} \bmod n = 1$ or $a^{2^{j-1} s} \bmod n = n - 1$.

If n is a prime then we wish to show that both these tests will always be true. Test 1 is equivalent to $a^{\phi(n)} \bmod n = 1$ (since $2^r s = n - 1 = \phi(n)$ for n prime), and this is exactly Fermat's little theorem. For this reason this test for primality is called the Fermat test. Test 2 is true since for every x, if $0 = (x^2 - 1) \bmod n = (x+1)(x-1) \bmod n$, then if n is prime we must have that either $(x+1) \bmod n = 0$ or $(x - 1) \bmod n = 0$. We thus obtain the following correctness theorem for witness:

$$\vdash\ \forall n, a.\ 0 < a < n \ \wedge\ \mathsf{witness}\ n\ a \ \Rightarrow\ \neg(\mathsf{prime}\ n) \quad (7)$$

2.2 Underlying Mathematics

A composite number n that passes a primality test for some base a is called a pseudoprime. In the case of the Fermat test, there exist numbers n that are

pseudoprimes for all bases a coprime to n. These numbers are called Carmichael numbers, and the two smallest examples are 561 and 1729.[5] Testing Carmichael numbers for primality using the Fermat test is just as hard as factorizing them, since the only bases that fail the test are multiples of divisors. Miller and Rabin's insight was that by also performing Test 2, the number of bases that are witnesses for any composite n will be at least $(n-1)/2$, as formalized in the following theorem:[6]

$$\begin{aligned}\vdash \forall\, n.&\\ &1 < n \ \wedge\ \mathsf{odd}\ n \ \wedge\ \neg(\mathsf{prime}\ n) \ \Rightarrow\\ &n - 1 \leq 2\left|\{a : 0 < a < n \ \wedge\ \mathsf{witness\ n\ a}\}\right| \end{aligned} \tag{8}$$

Therefore there are no Carmichael numbers for the Miller-Rabin test, and in fact just picking bases at random will quickly find a witness. This is the basis for the Miller-Rabin probabilistic primality test.

We now give a brief sketch of how Theorem 8 is proved, stating which classical results of number theory are necessary for the result.

The proof aims to find a proper subgroup B of the multiplicative group $\mathbb{Z}_n^*$ which contains all the nonwitnesses. This will then imply the result, since by Lagrange's theorem the size of a subgroup must divide the size of the group, and so $|B| \leq |\mathbb{Z}_n^*|/2 = \phi(n)/2 \leq (n-1)/2$.

If there exists an $x \in \mathbb{Z}_n^*$ such that $x^{n-1} \bmod n \neq 1$, then we choose $B = \{x \in \mathbb{Z}_n^* : x^{n-1} \bmod n = 1\}$. The Fermat test ensures that all nonwitnesses are members of B, and since B is closed under multiplication it is a proper subgroup of $\mathbb{Z}_n^*$. We may therefore assume that for every $x \in \mathbb{Z}_n^*$ we have that $x^{n-1} \bmod n = 1$.

Now suppose that $n = p^a$ is a prime power. In this case $\mathbb{Z}_n^*$ is cyclic, and so there exists an element $g \in \mathbb{Z}_n^*$ with order $\phi(n) = \phi(p^a) = p^{a-1}(p-1)$. But $g^{n-1} \bmod n = 1$, and so $p^{a-1}(p-1) \mid p^a - 1$. This is a contradiction, since $p \mid p^{a-1}(p-1)$ but $p \not| p^a - 1$, and so we may assume n is not a prime power.

We can therefore find two numbers $1 < a, b$ with $\mathsf{gcd}(a, b) = 1$ and $ab = n$. Next we find a maximal $j \in \{0, \ldots, r\}$ such that there exists a $v \in \mathbb{Z}_n^*$ with $v^{2^j s} \bmod n = n - 1$. Such a j must exist, because since s is odd we can set $j = 0$ and $v = n - 1$. Now choose

$$B = \{x \in \mathbb{Z}_n^* : x^{2^j s} \bmod n = 1 \ \vee\ x^{2^j s} \bmod n = n - 1\}$$

[5] 1729 is also famous as the Hardy-Ramanujan number, explained by C. P. Snow in the foreword to Hardy's *A Mathematician's Apology* [6]: "Once, in the taxi from London, Hardy noticed its number, 1729. He must have thought about it a little because he entered the room where Ramanujan lay in bed and, with scarcely a hello, blurted out his disappointment with it. It was, he declared, 'rather a dull number,' adding that he hoped that wasn't a bad omen. 'No, Hardy,' said Ramanujan, 'it is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways". $(10^3 + 9^3 = 1729 = 12^3 + 1^3)$

[6] In fact, it is possible to prove a stronger result that the number of witnesses must be at least $3\phi(n)/4$, and furthermore this bound is exact for the Carmichael number 8911.

B is closed under multiplication and so is a subgroup of $\mathbb{Z}_n^*$; also the maximality of j ensures that B must contain all nonwitnesses. It remains only to show that $B \neq \mathbb{Z}_n^*$. By the Chinese remainder theorem there exists $w \in \mathbb{Z}_n^*$ such that

$$w \bmod a = v \bmod a \ \wedge \ w \bmod a = 1$$

and so

$$w^{2^j s} \bmod a = a - 1 \ \wedge \ w^{2^j s} \bmod b = 1$$

Therefore by the Chinese remainder theorem $w^{2^j s} \bmod n$ is not equal to either 1 or $n - 1$, and so $w \notin B$ and the proof is complete.

2.3 Formalization

Formalizing this proof in HOL was a long but mostly routine task, resulting in the theories depicted in Fig. 1. The most time-consuming activity was a thorough development of group theory, from the initial axioms through to classical results such as Lagrange's theorem (9), Fermat's little theorem for groups (10) and the structure theorem for Abelian groups (11):

$$\vdash \forall G \in \mathsf{finite\_group}.\ \forall H \in \mathsf{subgroup}\ G.\ |\mathsf{set}\ H| \,\big|\, |\mathsf{set}\ G| \tag{9}$$

$$\vdash \forall G \in \mathsf{finite\_group}.\ \forall g \in \mathsf{set}\ G.\ g^{|\mathsf{set}\ G|} = e \tag{10}$$

$$\vdash \forall G \in \mathsf{finite\_group}.\ \mathsf{abelian}\ G \Rightarrow \exists g \in \mathsf{set}\ G.\ \forall h \in \mathsf{set}\ G.\ h^{|g|} = e \tag{11}$$

This development also allowed some classical arithmetic theorems to be expressed in the language of groups, including the Chinese remainder theorem (12) and the existence of primitive roots (13):

$$\begin{aligned}\vdash \forall p, q.\ & 1 < p \ \wedge \ 1 < q \ \wedge \ \mathsf{gcd}\ p\ q = 1 \ \Rightarrow \\ & (\lambda x.\ (x \bmod p, x \bmod q)) \in \\ & \quad \mathsf{group\_iso}\ (\mathsf{mult\_group}\ pq) \\ & \quad (\mathsf{prod\_group}\ (\mathsf{mult\_group}\ p)\ (\mathsf{mult\_group}\ q))\end{aligned} \tag{12}$$

$$\vdash \forall p, a.\ \mathsf{odd}\ p \ \wedge \ \mathsf{prime}\ p \ \wedge \ 0 < a \ \Rightarrow \ \mathsf{cyclic}\ (\mathsf{mult\_group}\ p^a) \tag{13}$$

As well as making the arithmetic theorems more concise, this also allowed the main proof to proceed entirely in the language of groups, eliminating the burden of switching mathematical context in the middle of a mechanical proof and incidentally mirroring the informal version of subsection 2.2.

The most difficult part of the whole formalization was Theorem 13 proving the existence of primitive roots. This required creating whole new theories of natural number polynomials and Abelian groups for the $a = 1$ case, and a subtle argument from Baker [1] for the step case.

One surprising difference between the informal mathematics and the formalization involved the use of the fundamental theorem of arithmetic. This states

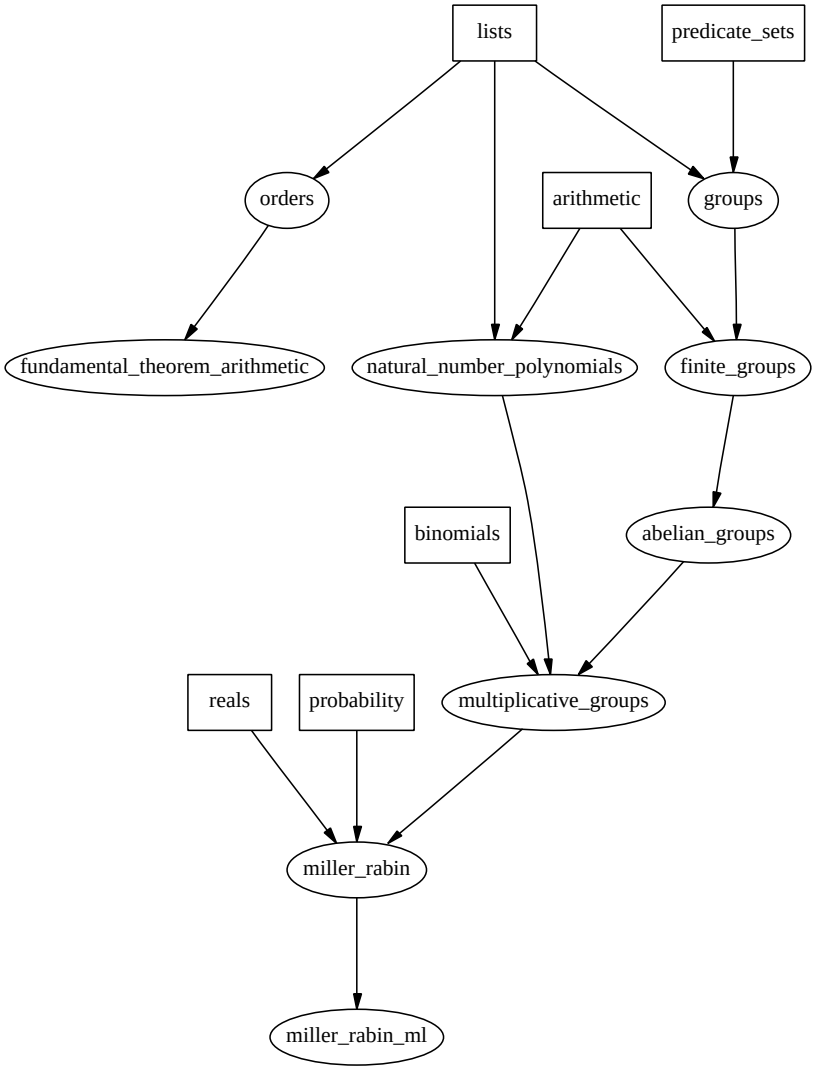

Fig. 1. The dependency relationships between the theories of the HOL formalization. Boxes indicate pre-existing HOL theories, and circles are theories created for this development.

that every natural number can be uniquely factorized into primes, and many informal mathematical proofs begin by applying this to some variable mentioned in the goal (e.g., by saying "let $p_1^{a_1} \cdots p_k^{a_k}$ be the prime factorization of n.") However, although we had previously formalized the fundamental theorem and it was ready to be applied, in the mechanical proofs we always chose what seemed to be an easier proof direction and so never needed it. Two examples of this phenomenom occur in the structure theorem for Abelian groups (11), and the cardinality of the witness set (8): the former theorem we formalized using least common multiples which proves the goal more directly; and in the latter all we needed was a case split between n being a prime power or being a product of coprime p, q, so we separately proved this lemma.

Finally, this development provided a testing ground for our predicate subtyping prover described in a recent paper [8]. Precisely, it was used as a condition prover in a contextual rewriter, and proved side-conditions such as group membership (e.g., $g *_G h \in \mathsf{set}\ G$), simple natural number inequalities (e.g., $0 < n$ or $1 < mn$) and nonemptiness properties of lists and sets (e.g., $s \neq \emptyset$).

Originally the aim was that more exotic properties could be proved by the predicate set prover, but it was found to be most robust on the relatively simple properties above that come up again and again during rewriting. These properties naturally propagate upwards through a term, being preserved by most of the basic operations, and in such situations the predicate set prover can be relied upon to show the desired condition (albeit sometimes rather slowly). This tool lent itself to more efficient development of the required theories, particularly the group theory where almost every theorem has one or more group membership side-conditions.

If our tool had not been available, it would have been possible to use the first-order prover to show most of the side-conditions, but there are three reasons why this is a less attractive proposition: firstly it would have required effort to find the right 'property propagation' theorems needed for the each goal; secondly the explicit invocations would have led to more complicated tactics; and thirdly some of the goals that can be proved using our specialized tool would simply have been out of range of a more general first-order prover.

3 Probability Theory

Recall that in our formalization of probability [7], we model probabilistic algorithms as state-transforming functions $\mathbb{B}^\infty \to \alpha \times \mathbb{B}^\infty$, where the state models the generator of random bits and has the type $\mathbb{B}^\infty$ of infinite boolean sequences.

In most algorithm textbooks this is how the Miller-Rabin test is defined:

> Given an odd integer n greater than 1, we pick a base a at random from the set $\{1, \ldots, n-1\}$ and call $\mathsf{witness}\ n\ a$. Suppose n is composite: since at least $(n-1)/2$ of the bases in the set are guaranteed to be witnesses, the probability that the procedure errs is at most $((n-1)/2)\,/\,(n-1) = 1/2$.

As mentioned in the introduction, this abstract view requires a generator of uniform random numbers, but most operating systems provide only a generator of

random bits. Furthermore, we showed in a previous paper [7] that a terminating algorithm to generate uniform random numbers in the range $\{0, \ldots, n-1\}$ from random bits does not exist unless n is a power of 2.[7] We therefore cannot directly use the simple version of Miller-Rabin above, but a single observation leads to a definition with the same probabilistic specification.

The observation is that the base 1 is always going to be a nonwitness for every n, so to find witnesses we can pick bases from the subset $\{2, \ldots, n-1\}$. Now if we can guarantee that the probability of picking each element from this subset is at least $1/(n-1)$, then the probability that we pick a witness is still at least $(1/(n-1))((n-1)/2) = 1/2$.

Using this observation relaxes the requirement for perfectly uniform random numbers, allowing any distribution that satisfies the lower bound. In our earlier work [7] we showed how to generate arbitrarily close approximations to uniform random numbers from random bits. This was done by introducing an extra parameter t, allowing us to prove the following theorem about the terminating algorithm uniform:

$$\vdash \ \forall t, n, k.\ k < n \ \Rightarrow \left|\mathbb{P}\left\{s : \mathsf{fst}\ (\mathsf{uniform}\ t\ n\ s) = k\right\} - 1/n\right| \leq 2^{-t} \tag{14}$$

Thus if we use the natural number function log2 that is related to calculating logarithms to the base 2

$$\vdash \forall n.\ \mathsf{log2}\ n = \mathsf{if}\ n = 0\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{suc}\ (\mathsf{log2}\ (n\ \mathrm{div}\ 2)) \tag{15}$$

$$\begin{aligned}\vdash \forall n, t.\ & \\ & 0 < n\ \wedge\ 2(\mathsf{log2}\ (n+1)) \leq t\ \Rightarrow \\ & 2^{-t} \leq 1/n - 1/(n+1)\end{aligned} \tag{16}$$

then the following theorem holds:

$$\begin{aligned}\vdash \forall t, n, k.\ & \\ & k < n\ \wedge\ 2(\mathsf{log2}\ (n+1)) \leq t\ \Rightarrow \\ & 1/(n+1) \leq \mathbb{P}\left\{s : \mathsf{fst}\ (\mathsf{uniform}\ t\ n\ s) = k\right\}\end{aligned} \tag{17}$$

Now we can define (one iteration of) the Miller-Rabin probabilistic primality test

$$\begin{aligned}\vdash \forall n, s.\ & \\ & \mathsf{miller\_rabin\_1}\ n\ s = \\ & \mathsf{if}\ n = 2\ \mathsf{then}\ (\top, s) \\ & \mathsf{else\ if}\ (n = 1)\ \vee\ \mathsf{even}\ n\ \mathsf{then}\ (\bot, s) \\ & \mathsf{else} \\ & \quad \mathsf{let}\ (a, s') \leftarrow \mathsf{uniform}\ (2(\mathsf{log2}\ (n-1)))\ (n-2)\ s \\ & \quad \mathsf{in}\ (\neg(\mathsf{witness}\ n\ (a+2)), s')\end{aligned} \tag{18}$$

[7] Here we are referring to guaranteed termination on every input sequence, not termination with probability 1.

satisfying the correctness theorems

$$\vdash \forall n, s.\ \mathsf{prime}\ n\ \Rightarrow\ \mathsf{fst}\ (\mathsf{miller\_rabin\_1}\ n\ s) = \top \tag{19}$$

$$\vdash \forall n.\ \neg(\mathsf{prime}\ n)\ \Rightarrow\ 1/2 \leq \mathbb{P}\{s : \mathsf{fst}\ (\mathsf{miller\_rabin\_1}\ n\ s) = \bot\} \tag{20}$$

$$\vdash \forall n.\ \mathsf{indep}\ (\mathsf{miller\_rabin\_1}\ n) \tag{21}$$

In order to define the full Miller-Rabin which tests several bases, we create a new (state-transformer) monadic operator many. The intention of many p n is a test that repeats n times the test p using different parts of the random bit stream, returning true if and only if each evaluation of p returned true. For instance, sdest is the destructor function for a stream, and so the function many sdest 10 tests that the next 10 booleans in the random stream are all $\top$. Here is the definition of many and basic properties:

$$\begin{aligned}\vdash\ & (\forall f.\ \mathsf{many}\ f\ 0 = \mathsf{unit}\ \top)\ \wedge \\ & (\forall f, n. \\ & \quad \mathsf{many}\ f\ (\mathsf{suc}\ n) = \mathsf{bind}\ f\ (\lambda x.\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{many}\ f\ n\ \mathsf{else}\ \mathsf{unit}\ \bot))\end{aligned} \tag{22}$$

$$\vdash \forall f, n.\ \mathsf{indep}\ f\ \Rightarrow\ \mathbb{P}\{s : \mathsf{fst}\ (\mathsf{many}\ f\ n\ s)\} = (\mathbb{P}\{s : \mathsf{fst}\ (f\ s)\})^n \tag{23}$$

$$\vdash \forall f, n.\ \mathsf{indep}\ f\ \Rightarrow\ \mathsf{indep}\ (\mathsf{many}\ f\ n) \tag{24}$$

Using the new many monadic operator it is simple to define the Miller-Rabin function miller_rabin

$$\vdash \forall n, t.\ \mathsf{miller\_rabin}\ n\ t = \mathsf{many}\ (\mathsf{miller\_rabin\_1}\ n)\ t \tag{25}$$

and finally Theorems 1 and 2 from the introduction follow from Theorems 19–24.

4 Extracting the Algorithm to Standard ML

The advantage of extracting the algorithm to a standard programming language such as ML is twofold: firstly execution is more efficient, and so the algorithm can be applied to usefully large numbers; and secondly it can be packaged up as a module and used as a reliable component of larger programs.

However, there is a danger that the properties that have been verified in the theorem-prover are no longer true in the new context. In this section we make a detailed examination of the following places where the change in context might potentially lead to problems: the source of random bits, the arbitrarily large natural numbers, and the manual translation of the Miller-Rabin functions to ML. Finally we test the algorithm on some examples, to check again that nothing has gone amiss and also to get some idea of the performance and computational complexity of the code.

4.1 Random Bits

Our theorems are founded on the assumption that our algorithms have access to a generator of perfectly random bits: each bit has probability of exactly $\frac{1}{2}$ of

being either 1 or 0, and is completely independent of every other bit. In the real world this idealized generator cannot exist, and we must necessarily select an approximation.

The first idea that might be considered is to use a pseudo-random number generator. These have been extensively analysed (for example by Knuth [10]) and pass many statistical tests for randomness, but their determinism makes them unsuitable for applications that require genuine unpredictability. For instance, when generating cryptographic keys it is not sufficient that the bits appear random, they must be truly unpredictable even by an adversary intent on exploiting the random number generator used.

Rejecting determinism, we must turn to the operating system for help. Many modern operating systems can utilize genuine non-determinism in the hardware to provide a higher quality of random bits. For example, here is a description of how random bits are derived and made available in Linux, excerpted from the relevant `man` page:

> "The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bit[s] of the noise in the entropy pool. From this entropy pool random numbers are created.
>
> When read, the `/dev/random` device will only return random bytes within the estimated number of bits of noise in the entropy pool. `/dev/random` should be suitable for uses that need very high quality randomness such as one-time pad or key generation. When the entropy pool is empty, reads to `/dev/random` will block until additional environmental noise is gathered.
>
> When read, `/dev/urandom` device will return as many bytes as are requested. As a result, if there is not sufficient entropy in the entropy pool, the returned values are theoretically vulnerable to a cryptographic attack on the algorithms used by the driver. Knowledge of how to do this is not available in the current non-classified literature, but it is theoretically possible that such an attack may exist. If this is a concern in your application, use `/dev/random` instead."

These devices represent the highest quality source of randomness to which we have easy access, and so we have packaged them up as ML boolean streams for use in our extracted program.

4.2 Arbitrarily Large Natural Numbers

Another place where there is a potential disparity between HOL and ML regards their treatment of numbers. The HOL algorithm operates on the natural numbers $\{0, 1, 2, \ldots\}$, while in ML the primitive type `int` contains signed numbers in a range depending on the machine architecture.

We resolved this incompatibility by creating the ML module `HolNum`, which implements an equality type `num` of arbitrarily large natural numbers. The Miller-Rabin functions may then use this type of numbers, and the arithmetic operations will behave exactly as in HOL.

We first implemented our own large number module, written in the purely functional subset of ML (and using a `word vector` representation of numbers). However, this was found to be about 100 times slower than the Moscow ML interface to the GNU Multi-Precision library, so we switched to this instead.

4.3 Extracting from HOL to ML

A further place where errors could creep in is the manual extraction of the Miller-Rabin functions from HOL to ML. Consequently we did this in two steps, the first of which was creating a new HOL theory containing a version of all the functions we wish to extract. For example, in Theorem 22 the monadic operator many was defined like so

$$\begin{aligned}\vdash\ & (\forall f.\ \mathsf{many}\ f\ 0 = \mathsf{unit}\ \top)\ \wedge \\ & (\forall f, n.\ \mathsf{many}\ f\ (\mathsf{suc}\ n) = \mathsf{bind}\ f\ (\lambda x.\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{many}\ f\ n\ \mathsf{else}\ \mathsf{unit}\ \bot))\end{aligned}$$

and in this new HOL theory we prove it is equivalent to

$$\begin{aligned}\vdash\ & \forall f, n. \\ & \quad \mathsf{many}\ f\ n = \\ & \quad \mathsf{if}\ n = 0\ \mathsf{then}\ \mathsf{unit}\ \top \\ & \quad \mathsf{else}\ \mathsf{bind}\ f\ (\lambda x.\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{many}\ f\ (n-1)\ \mathsf{else}\ \mathsf{unit}\ \bot)\end{aligned}$$

so that we may export it to ML as

```
fun MANY f n =
  if n = ZERO then UNIT true
  else BIND f (fn x => if x then MANY f (n -- ONE) else UNIT false);
```

As can be seen here the ML version involves some lexical changes, but has precisely the same parse tree as the intermediate HOL version. This reduces the chance of errors introduced by the cut-and-paste operation.

An intellectually interesting problem in the extraction is the question of how to handle partial functions. Consider the HOL function uniform that generates (approximations to) uniform random numbers:

$$\vdash \forall t, n.\ \mathsf{uniform}\ t\ (\mathsf{suc}\ n)\ s = \mathsf{if}\ t = 0\ \mathsf{then}\ \mathsf{unit}\ 0\ \mathsf{else}\ \ldots$$

The function is deliberately underspecified: there is no case where the second argument takes the value 0 because it does not make sense to talk of random numbers uniformly distributed over the empty set. HOL allows us to define functions like this, but there is no immediate ML equivalent. In the intermediate HOL version, we prove it to be equivalent to

$$\begin{aligned}\vdash\ & \forall t, n. \\ & \quad \mathsf{uniform}\ t\ n = \mathsf{if}\ n = 0\ \mathsf{then}\ \mathsf{uniform}\ t\ n\ \mathsf{else}\ \mathsf{if}\ t = 0\ \mathsf{then}\ \mathsf{unit}\ 0\ \mathsf{else}\ \ldots\end{aligned}$$

This rather strange theorem represents a total version of the uniform function. Of course if we simply extract this to ML it will loop forever when called with second argument 0, so we extract in the following way:

```
fun uniform t n =
  if n = ZERO then raise Fail "uniform: out of domain"
  else if t = ZERO then UNIT ZERO
  else ...
```

This device allows us to faithfully extract partial functions with as much confidence as for total functions.

4.4 Testing

It would be pleasant to say that since the function had been mechanically verified, no testing was necessary. But the preceding subsections have shown that this would be naive. Even if we are prepared to trust the generation of random bits, the operations of our arbitrarily large number module and the manual extraction of the algorithms, testing would still be prudent to catch bugs at the interface between these components.

We tested in the following way: for various values of l, generate n odd candidate numbers of length l bits. Perform a quick compositeness test on each by checking for divisibility by the first l primes, and also run Miller-Rabin with the maximum number of bases fixed at 50. The results are displayed in Table 1. $\mathbb{E}_{l,n}(\text{composite})$ is equal to $n(1 - \mathbb{P}_l(\text{prime}))$ and mathematically estimates the number of composites that the above algorithm will consider as candidates.[8] QC is the number of candidates that were found to be divisible by the quick compositeness test, MR is the number that the Miller-Rabin algorithm found to be composite, and finally MR+ is the number of candidates that the Miller-Rabin algorithm needed more than one iteration to determine that it was composite.

The most important property for testing purposes cannot be deduced from the table: for each number that the quick compositeness test found to be composite, the Miller-Rabin test also returned this result (and as can be seen, this almost always required only one iteration). In the other direction, using the $\mathbb{E}_{l,n}$ column as a guide, we can see that the Miller-Rabin algorithm did not find many more composites than expected.

In Table 2 we compare for each l the average time in seconds[9] taken to generate a random odd number, subject it to the quick composite test, and perform one iteration of the Miller-Rabin algorithm.

The complexity of (one iteration of) the Miller-Rabin algorithm is around $O(l^2 \log l)$, since it uses asymptotically the same number of operations as modular exponentiation (Cormen [4]). However, performing linear regression on the

[8] Using the prime number theorem, $\pi(n) \sim n/\log n$, we can derive:

$$\mathbb{P}_l(\text{prime}) = \frac{\pi(2^l) - \pi(2^{l-1})}{2^{l-2}} \sim \frac{2}{\log 2}\left(\frac{2}{l} - \frac{1}{l-1}\right)$$

[9] The results in this paper were produced using the Moscow ML 2.00 interpreter and RedHat Linux 6.2, running on a computer with a 200MHz Pentium Pro processor and 128Mb of RAM.

Table 1. Testing the Extracted Miller-Rabin Algorithm

l	n	$\mathbb{E}_{l,n}$	QC	MR	MR_+
10	100000	74352	70262	70683	520
15	100000	82138	72438	80448	85
20	100000	86332	74311	85338	5
50	100000	94347	79480	94172	0
100	100000	97144	82258	97134	0
150	100000	98089	83401	98077	0
200	100000	98565	84370	98557	0
500	100000	99424	86262	99458	0
1000	100000	99712	87377	99716	0
1500	100000	99808	87935	99798	0
2000	100000	99856	88342	99852	0

Table 2. Profiling the Extracted Miller-Rabin Algorithm

l	Gen time	QC time	MR_1 time
10	0.0004	0.0014	0.0028
15	0.0007	0.0017	0.0041
20	0.0009	0.0019	0.0054
50	0.0023	0.0034	0.0136
100	0.0068	0.0075	0.0370
150	0.0107	0.0112	0.0584
200	0.0157	0.0156	0.0844
500	0.0443	0.0416	0.2498
1000	0.0881	0.0976	0.7284
1500	0.1543	0.2164	1.7691
2000	0.3999	0.2843	4.2910

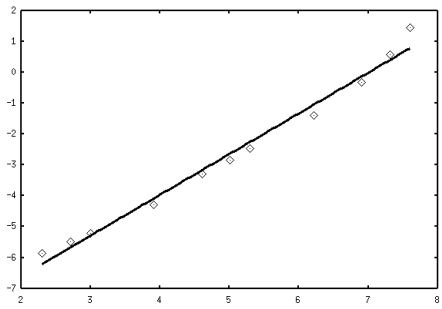

Fig. 2. Graph of $\log(\mathrm{MR}_1$ time) against $\log l$.

log-log graph in Fig. 2 gives a good fit with degree 1.32, implying a complexity of $O(l^{1.32})$. We can only conclude that the GNU Multi-Precision library is heavily optimized for the 'small' numbers in the range we were using, and so we cannot expect an asymptotically valid result.[10]

5 Conclusion

In this paper we have shown that our HOL probability theory is powerful enough to formally specify and verify the Miller-Rabin primality test, a well-known and commerically used probabilistic algorithm. The verification highlighted a small gap between theory and implementation, namely the difference between having a generator of uniformly distributed random numbers and a generator of random bits. An extra observation bridged this gap in the proof, and we were able to produce a version of Miller-Rabin using random bits that was guaranteed to terminate and satisfied the required probabilistic specification.

The predicate subtype condition prover helped to make the proof development more efficient, it was particularly useful for proving group membership conditions and simple but ubiquitous arithmetic conditions. Our evaluation is that it is a useful tool for reasoning about term properties that naturally propagate up terms. Overall, the proof script for the whole verification is 8000 lines long over 16 theories.

The difference between formal and informal proofs in their use of the fundamental theorem of arithmetic was pointed out in Subsection 2.3. This is the most striking example of many small differences in the style of informal and formal proofs, stemming from the different proof consumers in each case. Machines make it easier to formalize principles of induction such as dividing out a prime or prime power factor of a number, whereas humans would seem to be better at manipulating the multisets that contain the prime factors.

We have extracted an algorithm to Standard ML that takes as input a number and a stream of random bits, and declares the number either to be composite or probably prime, with a formally specified probability. Probabilistic algorithms such as these are difficult to get right since testing must necessarily be statistical, in this paper we have given arguments that our version has a high assurance of correctness.

6 Further Work

The verification of the Miller-Rabin algorithm is a fairly self-contained project, demonstrating the expressivity of our probability theory. However, there is work still to be done on the underlying tools and theories.

[10] When we ran this experiment using our own purely functional implementation of arbitrarily large numbers, it was a different story. Performing linear regression on the log-log graph gave a good fit with degree 2.98, confirming the theoretical result since we used the simple $O(l^2)$ algorithm for multiplication. Garbage collection was minimal, typically accounting for less than 5% of the time taken.

Firstly, since we now have a preliminary evaluation of the predicate subtype prover, we must decide the direction in which to advance it. On the one hand it may be possible to widen its effective scope by including rules for building up more complicated predicate sets, or on the other hand it may be most useful to speed up the performance and make it more robust on the scope we have identified in this paper. The path we take may depend on how far we can push the underlying higher-order prover.

Our probability theory is also ripe for development. Although it is general enough to capture any programs that use random bits, there are still new constants we can define to make formalization easier. An example in this paper was the many monadic operator, we are confident there will be others that will become apparent when we formalize more probabilistic algorithms.

Finally the extraction to ML was an interesting part of this project, and we built from scratch the infrastructure that allowed us to do this for our algorithm. It may be useful to push this further, perhaps aiming for a whole library of verified functions.

7 Related Work

There has been a long history of number theory formalizations, most relevantly for us: Russinoff's proof of Wilson's theorem in the Boyer-Moore theorem-prover [12]; Boyer and Moore's correctness proof of the RSA algorithm in ACL2 [2]; and Théry's correctness proof of RSA in three different theorem-provers [13]. This last work was especially useful, since one of the theorem-provers was hol98, and we were able to use his proof of the binomial theorem in our own development.

The closest work in spirit to this paper is Caprotti and Oostdijk's [3] primality proving in Coq, in which they formalize a similar computational number theory development and utilize a computer algebra system to prove numbers prime. Seeing this work improved the organization of theories in our own formalism. Harrison has also implemented a primality prover in HOL (Light), using Pratt's criterion instead of Pocklington's.

Our own development of group theory benefitted from the higher-order logic formalisms of Gunter [5], Kamueller [9] and Zammit [14], but the theory of groups has been formalized in many different theorem-provers.

There exist other formalizations of probability, and some have been applied to analysing probabilistic programs. It is conceivable that the same work could have been carried out using the probabilistic predicate transformers of Morgan et. al. [11], except that this formalism has not yet been mechanized.

Acknowledgements

This paper represents the fruits of a long-term project, and I would like to thank Judita Preiss, Michael Norrish, Konrad Slind, and my supervisor Mike

Gordon for their encouragement and advice throughout. Also David Preiss and Desmond Sheiham helped me to interpret some tricky textbook mathematics proofs; without their help the formalization of computational number theory would not have been so smooth.

References

1. Alan Baker. *A Concise Introduction to the Theory of Numbers*. Cambridge University Press, 1984.
2. Robert S. Boyer and J. Strother Moore. Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, 91(3):181–189, 1984.
3. O. Caprotti and M. Oostdijk. Formal and efficient primality proofs by use of computer algebra oracles. *Journal of Symbolic Computation*, 2001. Special Issue on Computer Algebra and Mechanized Reasoning (To appear).
4. Thomas H. Cormen, Charles Eric Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, Cambridge, Massachusetts, 1990.
5. Elsa Gunter. Doing algebra in simple type theory. Technical Report MS-CIS-89-38, Logic & Computation 09, Department of Computer and Information Science, University of Pennsylvania, 1989.
6. G. H. Hardy. *A Mathematician's Apology, reprinted with a foreword by C. P. Snow*. Cambridge University Press, 1993.
7. Joe Hurd. Lightweight probability theory for verification. Technical Report CSE-00-009, Oregon Graduate Institute, 2000.
8. Joe Hurd. Predicate subtyping with predicate sets. Accepted for TPHOLs 2001, February 2001.
9. F. Kammüller and L. C. Paulson. A formal proof of Sylow's first theorem – an experiment in abstract algebra with Isabelle HOL. *Journal of Automated Reasoning*, 23(3-4):235–264, 1999.
10. Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1997. Third edition.
11. Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. Probabilistic predicate transformers. Technical Report TR-4-95, Oxford University Computing Laboratory Programming Research Group, February 1995.
12. David M. Russinoff. An experiment with the Boyer-Moore theorem prover: A proof of Wilson's theorem. *Journal of Automated Reasoning*, 1:121–139, 1985.
13. Laurent Théry. A quick overview of PVS and HOL. Lecture Notes from the Types Summer School '99: Theory and Practice of Formal Proofs, Giens, France, August 1999.
14. Vincent Zammit. *On the Readability of Machine Checkable Formal Proofs*. PhD thesis, University of Kent at Canterbury, March 1999.