Verification of the Miller-Rabin Probabilistic Primality Test

Joe Hurd

University of Cambridge

1. Introduction
2. Computational Number Theory
3. Probabilistic Algorithms
4. Defining the Miller-Rabin Test
5. Extraction to ML
6. Conclusion

HOL on 1 Slide

We use the HOL theorem-prover, which is programmed in ML and implements higher-order logic. The terms of the logic are terms of Church's simply-typed lambda calculus (with Hindley-Milner polymorphism). Don't panic: most of our terms could be written in first-order logic.

The usual style of development in HOL is definitional extension, which means just making definitions and proving theorems. We **never** extend the initial axioms and rules of inference of the logic, and these remain the **only** way to create ML values of type `theorem`.

We can write arbitrary programs in ML to package up typical patterns of reasoning (tactics do this), but ultimately they must create theorems using the initial axioms and rules of inference. This is called fully-expansive proof.

Introduction

We define in HOL a probabilistic function miller_rabin and prove the following two properties of it:

$$\vdash \forall n, t, s.\ \mathsf{prime}\ n \Rightarrow \mathsf{fst}\ (\mathsf{miller_rabin}\ n\ t\ s) = \top$$

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

The Miller-Rabin algorithm is a probabilistic primality test, used by commercial software such as Mathematica.

Computational Number Theory

Let's plunge in and define an auxiliary function in higher-order logic:

$$\begin{array}{l} \vdash \quad \forall n. \\ \qquad \mathsf{factor_twos}\ n = \\ \qquad \mathsf{if}\ 0 < n\ \wedge\ \mathsf{even}\ n\ \mathsf{then} \\ \qquad\quad \mathsf{let}\ (r, s) \leftarrow \mathsf{factor_twos}\ (n \operatorname{div} 2) \\ \qquad\quad \mathsf{in}\ (\mathsf{suc}\ r, s) \\ \qquad \mathsf{else}\ (0, n) \end{array}$$

The HOL recursive definition package (called TFL) requires the user to prove the definition is well-formed (i.e., the function terminates), and behind the scenes makes a constant definition $\mathsf{factor_twos} = \cdots$ and derives the above theorem.

Computational Number Theory

Here's a HOL function to efficiently calculate modular exponentiations:

$$
\begin{aligned}
\vdash\ & \forall n, a, b. \\
& \quad \mathsf{modexp}\ n\ a\ b = \\
& \quad \mathsf{if}\ b = 0\ \mathsf{then}\ 1 \\
& \quad \mathsf{else} \\
& \quad\quad \mathsf{let}\ r \leftarrow \mathsf{modexp}\ n\ a\ (b\,\mathrm{div}\ 2) \\
& \quad\quad \mathsf{in\ let}\ r_2 \leftarrow (r^2 \bmod n) \\
& \quad\quad \mathsf{in\ if\ even}\ b\ \mathsf{then}\ r_2\ \mathsf{else}\ (r_2 a \bmod n)
\end{aligned}
$$

This uses repeated squaring, and requires $O(\log b)$ modular multiplications. The naive alternative of calculating a^b and then reducing modulo n is infeasible for large a, b, n.

Computational Number Theory

Here are the correctness theorems for the two auxiliary functions, derived from their definitions using the HOL rules of inference:

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

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

Note the (non-obvious) side-conditions that have appeared: verifying the functions in HOL has made explicit their assumptions.

Computational Number Theory

We next define a HOL function that inputs a possible prime n and a base a, calls factor_twos to find r, s such that s is odd and $2^r s = n - 1$, then uses modexp 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 $a^{2^{j-1} s} \bmod n = n - 1$.

Test 1 is the Fermat Test, and it is used by PGP to create public/private key pairs.

Computational Number Theory

Here is the definition of the HOL function; $\mathsf{witness}\ n\ a$ is true iff a is a 'witness' to the compositeness of n (i.e., the sequence from a causes n to fail one of the two primality tests).

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

Computational Number Theory

We can prove the following correctness theorem for witness:

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

This says that there will be no witnesses if n is a prime: good! But how many witnesses will there be if n is a composite?

If we just use the Fermat test, then there exist Carmichael numbers (e.g., 561, 1729) that have no witnesses except multiples of factors. Testing these numbers for primality using the Fermat test is just as hard as factoring them.

Are there Carmichael numbers for the Miller-Rabin test?

Computational Number Theory

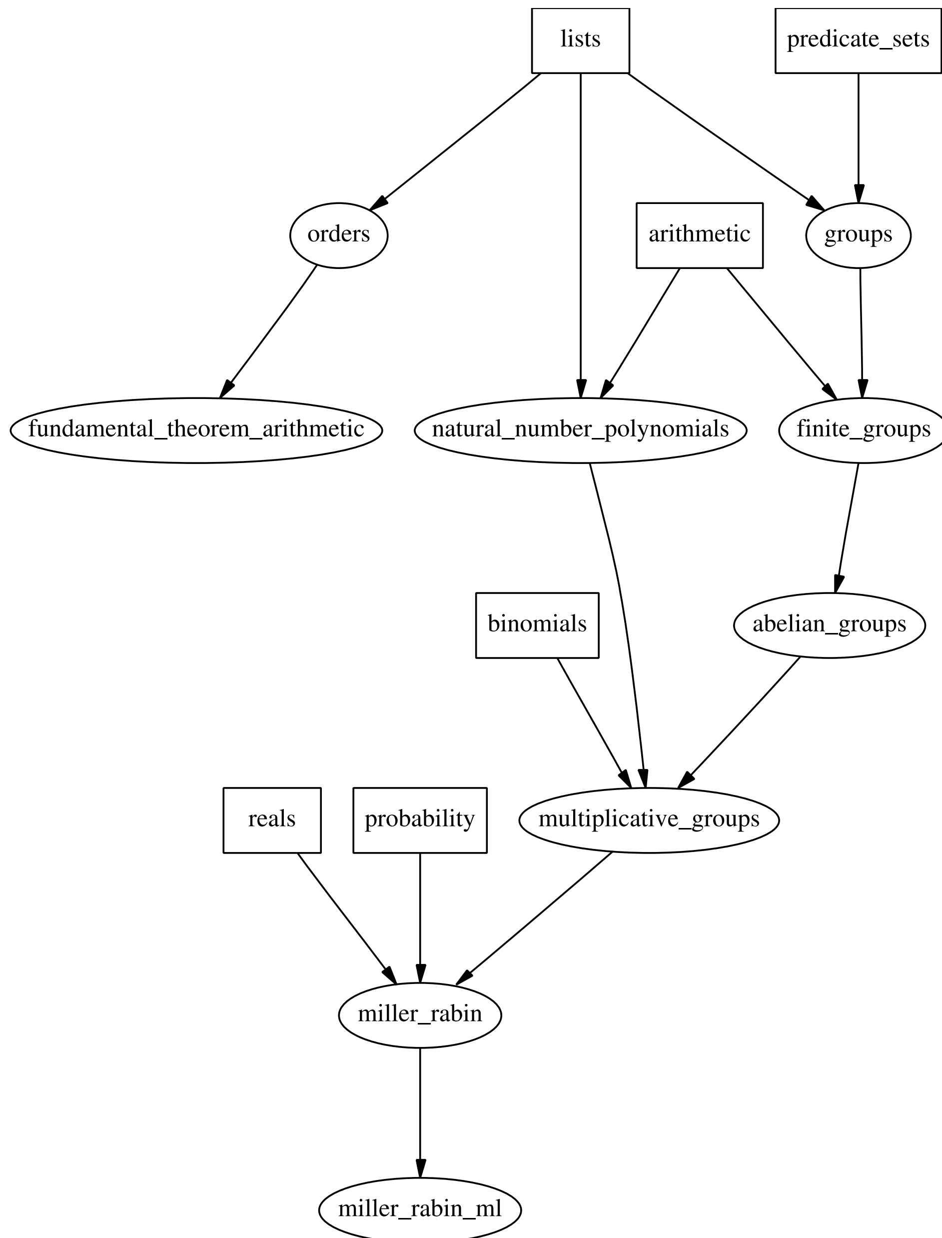
No. In fact, if n is composite then at least half of all possible bases a are witnesses:

$$\vdash \quad \forall n.\ 1 < n \ \wedge\ \mathsf{odd}\ n \ \wedge\ \neg(\mathsf{prime}\ n) \ \Rightarrow\ n - 1 \leq 2 \left|\{a : 0 < a < n \ \wedge\ \mathsf{witness\ n\ a}\}\right|$$

This means that picking bases a at random will quickly lead to a witness if n is composite, which forms the basis of the Miller-Rabin probabilistic primality test.

But first, before we show how to model probabilistic algorithms in HOL, how did we formalize the above theorem?

Formalization



Probabilistic Algorithms

We define in HOL a type $\mathbb{B}^\infty$ of infinite boolean sequences, and model a probabilistic function

$$f : \alpha \to \beta$$

with a corresponding deterministic function

$$F : \alpha \to \mathbb{B}^\infty \to \beta \times \mathbb{B}^\infty$$

This method of 'passing around the random-number generator' is also used in pure functional languages such as Haskell, and allows an elegant formulation of probabilistic programs in terms of state transforming monads:

$$\begin{aligned} \mathsf{unit}\ x &= \lambda\, s.\ (x, s) \\ \mathsf{bind}\ f\ g &= (\lambda\, (x, s).\ g\ x\ s) \circ f \end{aligned}$$

Probability Theory

We build upon Harrison's construction of the real numbers; adding ingredients from mathematical measure theory to allow the essential concepts of probability and independence to be defined.

We have formalized in HOL a theory of finite probability, and proved results such as the following theorem:

Thm: If a probabilistic program can be constructed using our monadic primitives, then the returned value is independent of the returned sequence.

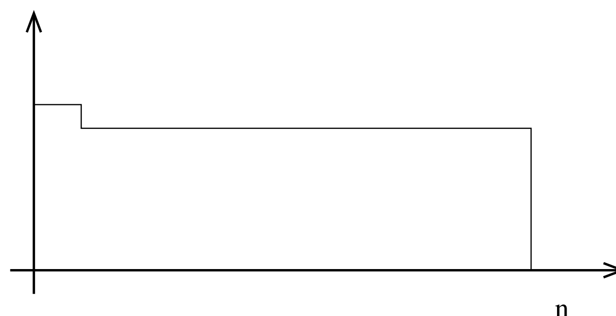
Note: the converse is not true: $\lambda s.\ (s_0 = s_1, \mathsf{stl}\ s)$

This indicates how tricky independence can be.

The Miller-Rabin Function

Most presentations of the Miller-Rabin in algorithm textbooks assume a generator of perfectly uniform random numbers in the range $\{0, \ldots, n-1\}$, but this this cannot be implemented by any terminating algorithm using random bits (unless n is a power of 2).

Instead we define a HOL function that generates numbers with an approximately uniform distribution. We pass in an extra parameter t, and the probability of $\mathsf{uniform}\ t\ n$ returning each number in the range is within 2^{-t} of $1/n$.



The Miller-Rabin Function

How can we use our approximation function uniform to define a Miller-Rabin function?

If we observe that 1 will never be a witness for any n, then 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.

The Miller-Rabin Function

If we define the unif_bound function (related to the $\log_2$ function)

$$\vdash \quad \forall n.\ \mathsf{unif_bound}\ n = \mathsf{if}\ n = 0\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{suc}\ (\mathsf{unif_bound}\ (n \operatorname{div} 2))$$

then the desired lower bound can be achieved:

$$\vdash \quad \forall t, n, k.\ k < n\ \wedge\ 2(\mathsf{unif_bound}\ (n+1)) \leq t\ \Rightarrow 1/(n+1) \leq \mathbb{P}\,\{s : \mathsf{fst}\ (\mathsf{uniform}\ t\ n\ s) = k\}$$

The Miller-Rabin Function

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

$$
\begin{array}{l}
\vdash \quad \forall n, s. \\
\qquad \mathsf{miller_rabin_1}\ n\ s = \\
\qquad \mathsf{if}\ n = 2\ \mathsf{then}\ (\top, s) \\
\qquad \mathsf{else\ if}\ (n = 1)\ \vee\ \mathsf{even}\ n\ \mathsf{then}\ (\bot, s) \\
\qquad \mathsf{else} \\
\qquad\quad \mathsf{let}\ (a, s') \leftarrow \\
\qquad\qquad \mathsf{uniform}\ (2(\mathsf{unif_bound}\ (n-1)))\ (n-2)\ s \\
\qquad\quad \mathsf{in}\ (\neg(\mathsf{witness}\ n\ (a+2)), s')
\end{array}
$$

The Miller-Rabin Function

This satisfies the correctness theorems

$$\vdash \quad \forall\, n, s.\ \mathsf{prime}\ n \ \Rightarrow\ \mathsf{fst}\ (\mathsf{miller_rabin_1}\ n\ s) = \top$$

$$\vdash \quad \forall\, n.$$
$$\neg(\mathsf{prime}\ n) \ \Rightarrow$$
$$1/2 \le \mathbb{P}\,\{s : \mathsf{fst}\ (\mathsf{miller_rabin_1}\ n\ s) = \bot\}$$

$$\vdash \quad \forall\, n.\ \mathsf{indep}\ (\mathsf{miller_rabin_1}\ n)$$

We now need to define the full Miller-Rabin algorithm, which involves independently choosing many bases and checking each one. The last correctness theorem above will be critical for this.

The Miller-Rabin Function

Instead of directly defining miller_rabin, we make one last generalization: the many monadic operator.

$$
\begin{aligned}
\vdash\ & (\forall f.\ \mathsf{many}\ f\ 0 = \mathsf{unit}\ \top)\ \wedge \\
& (\forall f, n. \\
& \quad \mathsf{many}\ f\ (\mathsf{suc}\ n) = \\
& \quad \mathsf{bind}\ f\ (\lambda x.\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{many}\ f\ n\ \mathsf{else}\ \mathsf{unit}\ \bot)) \\
\vdash\ & \forall f, n. \\
& \quad \mathsf{indep}\ f\ \Rightarrow \\
& \quad \mathbb{P}\,\{s : \mathsf{fst}\ (\mathsf{many}\ f\ n\ s)\} = (\mathbb{P}\,\{s : \mathsf{fst}\ (f\ s)\})^n \\
\vdash\ & \forall f, n.\ \mathsf{indep}\ f\ \Rightarrow\ \mathsf{indep}\ (\mathsf{many}\ f\ n)
\end{aligned}
$$

Note that like many preserves independence, and so can be used as a building block in more complicated probabilistic programs.

The Real Miller-Rabin Function

Finally, we may now define the function we want

$$\vdash \quad \forall\, n, t.\ \mathsf{miller_rabin}\ n\ t = \mathsf{many}\ (\mathsf{miller_rabin_1}\ n)\ t$$

and the desired properties from the introduction follow immediately:

$$\vdash \quad \forall\, n, t, s.\quad \mathsf{prime}\ n \;\Rightarrow\; \mathsf{fst}\ (\mathsf{miller_rabin}\ n\ t\ s) = \top$$

$$\vdash \quad \forall\, n, t.\quad \neg(\mathsf{prime}\ n) \;\Rightarrow\; 1 - 2^{-t} \le \mathbb{P}\,\{s : \mathsf{fst}\ (\mathsf{miller_rabin}\ n\ t\ s) = \bot\}$$

Now we have proved the implementation correct, can we execute it?

Extraction to Standard ML

Yes! By extracting the function from HOL to ML, but we must be careful to preserve the context in which the function was verified.

1. How can we get a source of perfectly random bits?
2. Our HOL algorithm uses arbitrarily large natural numbers, but the default ML integer type is signed and fixed width.
3. How can we avoid cut-and-paste errors in the manual extraction from HOL to ML?

Even if we can adequately solve these component problems, we still must test the resulting program to find bugs at the interfaces.

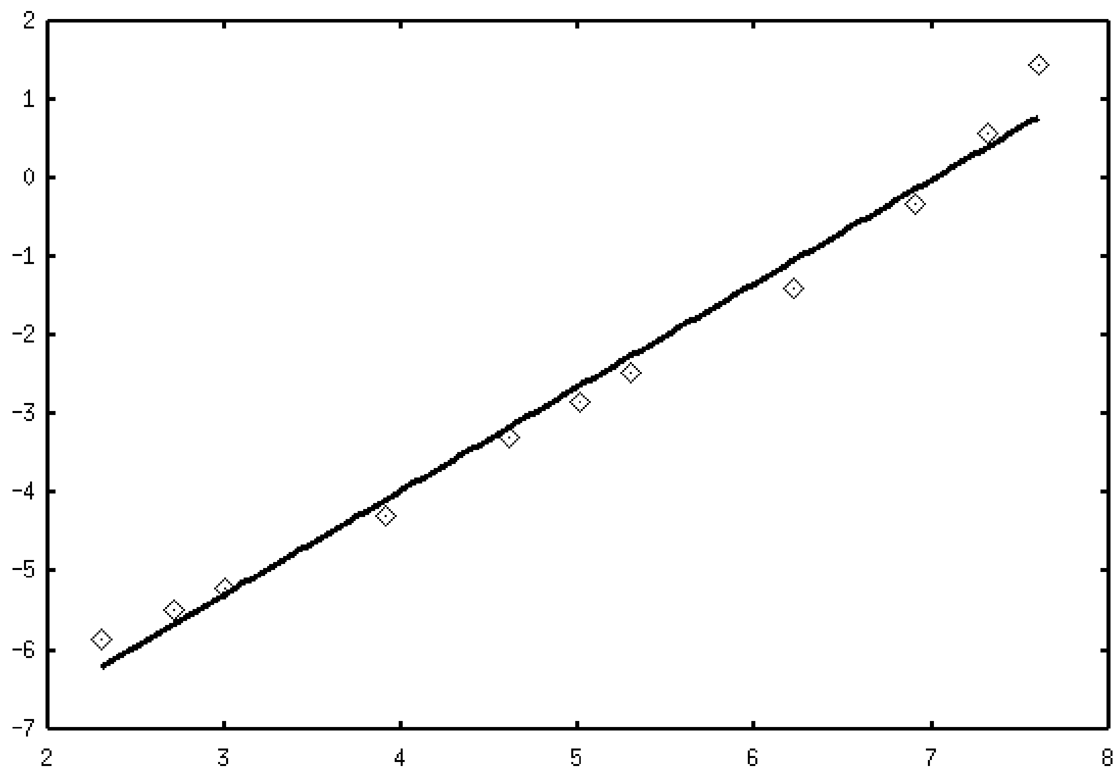
Performance

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

Performance

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

Performance



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

Line of best fit is $y = 1.32x - 9.24$, so from our experiments MR_1 time is $O(l^{1.32})$.

Cormen, Leiserson and Rivest prove it to be asymptotically the same as modular exponentiation ($\approx O(l^2 \log l)$).

Conclusions

We have demonstrated that our HOL probability theory is powerful enough to formally specify and verify the Miller-Rabin primality test, a well-known and commercially used probabilistic algorithm.

We showed how to implement the Miller-Rabin algorithm using a generator of random bits instead of perfect uniformly distributed random numbers, while preserving correctness.

Finally, we extracted the Miller-Rabin algorithm to Standard ML, making available to programmers an implementation with a high assurance of correctness.

Note: the full paper is available from my website:

`http://www.cl.cam.ac.uk/~jeh1004`