Lightweight Probability Theory for Verification

Joe Hurd

University of Cambridge

1. Motivation
2. A Language for Probabilistic Algorithms
3. Formalizing Probability Theory
4. A Uniform Random Number Generator

Motivation

The Miller-Rabin primality test takes a number n and returns either PRIME or COMPOSITE. If n actually is prime then it is guaranteed to return PRIME, and if n is composite then it will return COMPOSITE with probability at least one half. Successive calls are independent, so if n is composite then s consecutive results of PRIME will occur with probability at most 2^{-s}.

How can we specify and verify such an algorithm?

To answer this question, we have created the following two theories in HOL:

- A language for expressing probabilistic programs.
- A formalization of (basic) probability theory.

A Language for Probabilistic Algorithms

The programming language we use is the language of higher-order logic functions.

We define 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.

Formalizing 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. This results in a lightweight probability theory.

This leads to an important result:

Thm: For all probabilistic programs constructed using our monadic primitives (including Haskell probabilistic programs), the returned value is independent of the returned sequence.

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

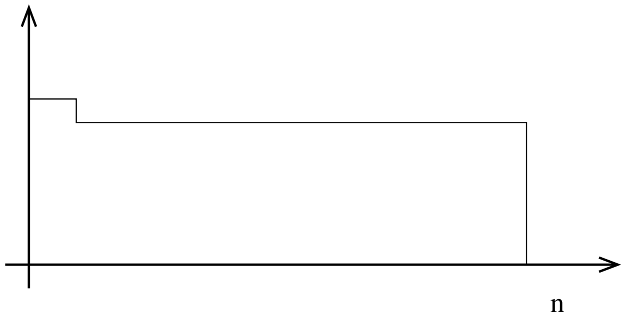
This indicates how tricky independence can be.

A Uniform Random Number Generator

We made use of this development to write a probabilistic function that returned random numbers in the range $0, 1, \ldots, n-1$.

We originally wanted the returned numbers to be uniformly distributed on the range, but this turns out to be impossible unless n is a power of two!

We settled for almost-uniform:



We pass in an extra parameter t, and the probability of returning each number in the range is within 2^{-t} of $1/n$.