Formalization of the Standard Uniform Random Variable in HOL

Osman Hasan and Sofiène Tahar

Department of Electrical and Computer Engineering,
Concordia University, Montreal, Canada
Email: {o_hasan, tahar}@ece.concordia.ca

Technical Report

December, 2006

Abstract

Continuous random variables are widely used to mathematically describe random phenomenon in engineering and physical sciences. In this paper, we present a higher-order logic formalization of the Standard Uniform random variable. We show the correctness of this specification by proving the corresponding probability distribution properties within the HOL theorem prover and the proof steps have been summarized. This formalized Standard Uniform random variable can be transformed to formalize other continuous random variables, such as Uniform, Exponential, Normal, etc., by using various Non-uniform random number generation techniques. The formalization of these continuous random variables will enable us to perform error free probabilistic analysis of systems within the framework of a higher-order-logic theorem prover. For illustration purposes, we present the formalization of the Continuous Uniform random variable based on our Standard Uniform random variable and then utilize it to perform a simple probabilistic analysis of roundoff error in HOL.

1 Introduction

Probabilistic analysis has become a tool of fundamental importance to virtually all scientists, engineers, medical practitioners, jurists and industrialists. Consider the example of modern engineering. It deals with the design and construction of devices that exhibit significant random or unpredictable elements. Moreover, these devices act upon and within complex environments that themselves have significant elements of unpredictability. The engineering and physical science approach to mastering the unavoidable elements of randomness and uncertainty is to use random variables to mathematically describe random phenomenon. Then, the system performance and reliability issues are judged based on the corresponding probabilistic and statistical properties.

Random variables are basically functions that map random events to numbers. Every random variable gives rise to a probability distribution, which contains most of the important information about this random variable. The probability distribution of a random variable X can be uniquely described by its *cumulative distribution function* (CDF), which is defined by $F_X(x) = \mathbb{P}(X \leq x)$ for any real number x, where $\mathbb{P}$ represents the probability. A distribution is called discrete if its CDF consists of a sequence of finite jumps, which means that it belongs to a random variable that can only attain values from a certain finite or countable set. Similarly, a distribution is called continuous if its CDF is continuous, which means that it belongs to a random variable that ranges over a continuous set of real numbers. A continuous set of real numbers, sometimes referred to as an interval, contains all real numbers between two limits. An interval can be open (a,b) corresponding to the set $\{x|a < x < b\}$, closed [a,b] which represents the set $\{x|a \leq x \leq b\}$, or half-open (a,b], [a,b). Continuous probability distributions are widely used to mathematically describe random phenomenon in engineering and scientific applications. For example, Continuous Uniform distribution is used to model round-off errors [28], Exponential distribution occurs in applications such as queuing theory to model *interarrival* and *service* times and Normal distribution is extensively used to model signals in data transmission and DSP systems [27].

Today, simulation is the most commonly used computer based probabilistic analysis technique. Most simulation softwares provide a programming environment for defining functions that approximate random variables for probability distributions. The random elements in a given system are modeled by these functions and the system is analyzed using computer simulation techniques [8], such as the Monte Carlo Method [19], where the main idea is to approximately answer a query on a probability distribution by analyzing a large number of samples. Due to these approximations the results can be quite unreliable at times. It is a common occurrence that different software packages come up with different solutions to the same probabilistic problem. This unreliability mainly arises because of three major reasons. First, the primary source of randomness in these simulation based packages is pseudorandom numbers [17]. Secondly, functions for evaluating probability distributions are approximated using a variety of efficient but less accurate numerical methods [16]. Finally, like all computer based computations, roundoff and truncation errors also creep into the numerical computations conducted in these simulation based software packages. McCullough [20] proposed a collection of intermediate-level tests for assessing the numerical reliability of a statistical package and uncovered flaws in most of the mainframe statistical packages, e.g., [21]. These unreliable results pose a serious problem in safety critical applications, such as space travel, military and medicine, where a mismatch between the predicted and the actual system performance may result in either inefficient usage of the available resources or paying higher costs to meet some performance or reliability criteria unnecessarily. Another major limitation of simulation based probabilistic analysis is the enormous amount of CPU time requirement for attaining meaningful estimates. This approach generally requires hundreds of thousands of simulations to calculate the probabilistic quantities and becomes impractical when each simulation involves extensive computations.

As an alternative to simulation techniques, we propose to use higher-order logic interactive theorem proving [10] for probabilistic analysis. Higher-order logic is a system of deduction with a precise semantics and can be used for the development of almost all classical mathematics theories. Interactive theorem proving is the field of computer science and mathematical logic concerned with computer based formal proof tools that require some sort of human assistance. We believe that probabilistic analysis can be performed by specifying the behavior of systems which exhibit randomness in higher-order logic and formally proving the intended probabilistic properties within the environment of an interactive theorem prover. The probabilistic analysis carried out in this way will be free from approximation and precision issues, and the accuracy of the results will be independent of the CPU time.

The foremost criteria for implementing a formalized probabilistic analysis framework is to be able to formalize random variables in higher-order logic. Hurd's PhD thesis [15] can be considered a pioneering work in this regard as it presents a methodology for the formalization of probabilistic algorithms in the higher-order-logic (HOL) theorem prover [11]. Random variables are basically probabilistic algorithms and Hurd formalized some discrete random variables in [15]. On the other hand, Hurd's methodology cannot be used, as is, to formalize continuous random variables. In fact, to the best of our knowledge, no higher-order-logic formalization of continuous random variables exists in the literature so far. In this paper, we show how to extend Hurd's formalization framework for the formalization of continuous random variables using Non-uniform random number generation methods [8]. The main advantage of this approach is that only one continuous random variable needs to be formalized, i.e., the Standard Uniform random variable. Other continuous random variables can then be modeled by formalizing the corresponding Non-uniform random number generation method.

The rest of the paper is organized as follows: In Section 2, we provide some preliminaries including a brief introduction to the HOL theorem prover and an overview of Hurd's methodology for the formalization of probabilistic algorithms in HOL. In Section 3, we formally specify the Standard Uniform random variable in the HOL theorem prover as the limit value of a discrete uniform random variable in the interval $[0, 1 - (\frac{1}{2})^n]$. In Section 4, we verify the correctness of the above specification by proving its corresponding probability distribution properties. In order to illustrate the fact that the formalized Standard Uniform random variable can be utilized to formalize other continuous random variables using Non-uniform random number methods, we formally specify and verify the Continuous Uniform random variable in Section 5. Then in Section 6, we highlight the practical effectiveness of the proposed approach by comparing the results of simulation based and the proposed theorem proving based approaches for a very simple probabilistic analysis example of roundoff error in a digital processor. We present a review of related work in literature in Section 7 and finally conclude the paper in Section 8.

2 Preliminaries

In this section we give a brief introduction to the HOL theorem prover and present an overview of Hurd's methodology for the formalization of probabilistic algorithms proposed in [15]. The intent is to introduce the main ideas along with some notation that is going to be used in the coming sections.

2.1 HOL Theorem Prover

The HOL theorem prover is an interactive theorem prover which is capable of conducting proofs in higher-order logic. It utilizes the simple type theory of Church [5] along with Hindley-Milner polymorphism [22] to implement higher-order logic. HOL has been successfully used as a verification framework for both software and hardware as well as a platform for the formalization of pure mathematics. It supports the formalization of various mathematical theories including sets, natural numbers, real numbers, measure and probability. HOL is an interactive theorem prover with access to many proof assistants and automatic proof procedures. The user interacts with a proof editor and provides it with the necessary tactics to prove goals while some of the proof steps are solved automatically by the automatic proof procedures.

In order to ensure secure theorem proving, the logic in the HOL system is represented in the strongly-typed functional programming language ML [24]. The ML abstract data types are then used to represent higher-order-logic theorems and the only way to interact with the theorem prover is by executing ML procedures that operate on values of these data types. Users can prove theorems using a natural deduction style by applying inference rules to axioms or previously generated theorems. The HOL core consists of only 5 axioms and 8 primitive inference rules, which are implemented as ML functions. Soundness is assured as every new theorem must eventually be created from the 5 axioms or any other pre-existing theorems and the 8 primitive inference rules.

We selected HOL theorem prover for the proposed formalization mainly because of its ability to handle higher-order logic, inherent soundness and in order to benefit from the inbuilt mathematical theories for measure and probability. The table below summarizes some of the HOL symbols used in this paper and their corresponding mathematical interpretation [11].

HOL Symbol	**Standard Symbol**	**Meaning**
$\wedge$	*and*	Logical *and*
$\vee$	*or*	Logical *or*
num	$\{0, 1, 2, \ldots\}$	Natural data type
real	All Real numbers	Real data type
SUC n	$n + 1$	Successor of a *num*
$\lambda x.t$	$\lambda x.t$	Function that maps x to $t(x)$
$\{x \vert P(x)\}$	$\{\lambda x.P(x)\}$	Set of all x that satisfy the condition P
(a, b)	a x b	A mathematical pair of two elements

Table 1: HOL Symbols

2.2 Verifying Probabilistic Algorithms in HOL

Hurd [15] proposed to formalize the probabilistic algorithms in higher-order logic by thinking of them as deterministic functions with access to an infinite Boolean sequence $\mathbb{B}^\infty$; a source of infinite random bits. These deterministic functions make random choices based on the result of popping the top most bit in the infinite Boolean sequence and may pop as many random bits as they need for their computation. When the algorithms terminate, they return the result along with the remaining portion of the infinite Boolean sequence to be used by other programs. Thus, a probabilistic algorithm which takes a parameter of type α and ranges over values of type β can be represented in HOL by the function

$$\mathcal{F} : \alpha \rightarrow B^\infty \rightarrow \beta \times B^\infty$$

For example, a $Bernoulli(\frac{1}{2})$ random variable that returns 1 or 0 with equal probability $\frac{1}{2}$ can be modeled as follows

```
⊢ bit = λs.  (if shd s then 1 else 0, stl s)
```

where s is the infinite Boolean sequence and `shd` and `stl` are the sequence equivalents of the list operation *'head'* and *'tail'*. The function *bit* accepts the infinite Boolean sequence and returns a random number, which is either 0 or 1 together with a sequence of unused Boolean sequence, which in this case is the tail of the sequence. The above methodology can be used to model most probabilistic algorithms. All probabilistic algorithms that compute a finite number of values equal to 2^n, each having a probability of the form $\frac{m}{2^n}$: where m represents the hol data type *nat* and is always less than 2^n, can be modeled using well-founded recursion. The probabilistic algorithms that do not satisfy the above conditions but are sure to terminate can be modeled by the *probabilistic while loop* proposed in [15].

The probabilistic programs can also be expressed in the more general state-transforming monad where the states are the infinite Boolean sequences.

```
⊢ ∀ a,s.  unit a s = (a,s)
⊢ ∀ f,g,s.  bind f g s = let (x,s')← f(s) in g x s'
```

The `unit` operator is used to lift values to the monad, and the `bind` is the monadic analogue of function application. All the monad laws hold for this definition, and the notation allows us to write functions without explicitly mentioning the sequence that is passed around, e.g., function *bit* can be defined as

```
⊢ bit_monad =
        bind sdest (λb.  if b then unit 1 else unit 0)
```

where `sdest` gives the head and tail of a sequence as a pair (*shd* `s`, *stl* `s`).

Hurd [15] also formalized some mathematical measure theory in HOL in order to define a probability function $\mathbb{P}$ from sets of infinite Boolean sequences to real numbers between 0 and 1. The domain of $\mathbb{P}$ is the set $\mathcal{E}$ of events of the probability. Both $\mathbb{P}$ and $\mathcal{E}$ are defined using the Carathéodory's Extension theorem, which ensures that $\mathcal{E}$ is a σ-algebra: closed under complements and countable unions. The formalized $\mathbb{P}$ and $\mathcal{E}$ can be used to derive the basic laws of probability in the HOL prover, e,g., the additive law, which represents the probability of two disjoint events as the sum of their probabilities:

$\vdash \forall$ A B. A $\in \mathcal{E} \wedge$ B $\in \mathcal{E} \wedge$ A $\cap$ B = $\emptyset \Rightarrow$
$\mathbb{P}$(A $\cup$ B) = $\mathbb{P}$(A) + $\mathbb{P}$(B)

The formalized $\mathbb{P}$ and $\mathcal{E}$ can also be used to prove probabilistic properties for probabilistic programs such as

$\vdash \mathbb{P}$ {s | fst (bit s) = 1} = $\frac{1}{2}$

where the function fst selects the first component of a pair.

The measurability of a function is an important concept in probability theory and also a useful practical tool for proving that sets are measurable [3]. In Hurd's formalization of probability theory, a set of infinite Boolean sequences, S, is said to be measurable if and only if it is in $\mathcal{E}$, i.e., S $\in \mathcal{E}$. Since the probability measure $\mathbb{P}$ is only defined on sets in $\mathcal{E}$, it is very important to prove that sets that arise in verification really are measurable. Hurd [15] showed that if a function accesses the infinite boolean sequence using only the unit, bind and sdest primitives then the function is guaranteed to be measurable and thus leads to only measurable sets.

Hurd formalized four discrete probability distributions and proved their correctness by proving the corresponding PMF properties [15]. Because of their discrete nature, all these random variables either compute a finite number of values or are sure to terminate. Thus, they can be expressed using Hurd's methodology by either well formed recursive functions or the probabilistic while loop [15]. On the other hand, continuous random variables always compute an infinite number of values and therefore would require all the random bits in the infinite Boolean sequence if they are to be represented using Hurd's methodology. The corresponding deterministic functions cannot be expressed by either recursive functions or the probabilistic while loop and it is mainly for this reason that the specification of continuous random variables needs to be handled differently than their discrete counterparts.

3 Specification of Standard Uniform Random Variable in HOL

In this section, we present a formal specification of the Standard Uniform random variable in HOL. Standard Uniform random variable can be formalized in a number of different ways using the various formal semantics of probabilistic programs available in the literature of theoretical computer science. In order to minimize the effort and speed up the formalization process, our intent is to find a solution that enables us to build upon the existing work of Hurd [15].

Standard Uniform random variable is a continuous random variable for which the probability that it will belong to a subinterval of [0,1] is proportional to the length of that subinterval. It can be characterized by the CDF as follows:

$$\mathbb{P}(X \leq x) = \begin{cases} 0 & \text{if } x < 0; \\ x & \text{if } 0 \leq x < 1; \\ 1 & \text{if } 1 \leq x. \end{cases} \tag{1}$$

It is a well known mathematical fact, see [9] for example, that a Standard Uniform random variate can be modeled by an infinite sequence of random bits (informally coin flips) as follows

$$\sum_{k=0}^{\infty}(\frac{1}{2})^{k+1}X_k \tag{2}$$

where, X_k denotes the outcome of the k^{th} random bit; $true$ or $false$ represented as 1 or 0 respectively. The mathematical relation of Equation (2) presents a sampling algorithm for the Standard Uniform random variable which is quite consistent with Hurd's formalization methodology, i.e, it allows us to model the Standard Uniform random variable by a deterministic function with access to the infinite Boolean sequence. The specification of this sampling algorithm in higher-order logic is not very straight forward though. Due to the infinite sampling, it cannot be modeled by either of the approaches proposed in [15], i.e., a recursive function or the probabilistic while loop. We approach this problem by splitting the corresponding mathematical relation into two steps. The first step is to mathematically represent a discrete version of the Standard Uniform random variable

$$(\lambda n. \sum_{k=0}^{n-1}(\frac{1}{2})^{k+1}X_k) \tag{3}$$

This lambda abstraction function accepts a natural number n and generates an n-bit Standard Uniform random variable using the computation principle of Equation (2). The continuous Standard Uniform random variable can now be represented as a special case of Equation (3) when n tends to infinity

$$\lim_{n\to\infty}(\lambda n. \sum_{k=0}^{n-1}(\frac{1}{2})^{k+1}X_k) \tag{4}$$

The advantage of expressing the sampling algorithm of Equation (2) in these two steps is that now it can be specified in HOL. The mathematical relationship of Equation (3) can be specified in HOL by a recursive function using Hurd's methodology as it consumes a finite number of random bits, i.e., n. Then, the formalization of the mathematical concept of limit of a real sequence [13] in HOL can be used to specify the mathematical relation of Equation (4).

Next, we present the HOL formalization of the above steps. We first formalize a discrete Standard Uniform random variable that produces any one of the equally spaced 2^n dyadic rationals in the interval $[0, 1 - (\frac{1}{2})^n]$ with the same probability $(\frac{1}{2})^n$. This random variable is specified in HOL as a recursive function using Hurd's formalization framework of Section 2.2.

```
⊢ (std_unif_disc 0 = unit 0) ∧
     ∀ n.  (std_unif_disc (suc n) =
     bind (std_unif_disc n)
          (λm.  bind sdest
          (λb.  unit (if b then (($\frac{1}{2}$)$^{n+1}$ + m) else m))))
```

The function, *std_unif_disc*, models an n-bit discrete Standard Uniform random variable based on the principle of Equation (2) by simply converting the first n random bits $B_0, B_1, B_2, \ldots B_{n-1}$ of the infinite Boolean sequence to their equivalent real number with the binary representation $0.B_0B_1B_2 \ldots B_{n-1}$. It returns a pair such that the first component is the n-bit discrete

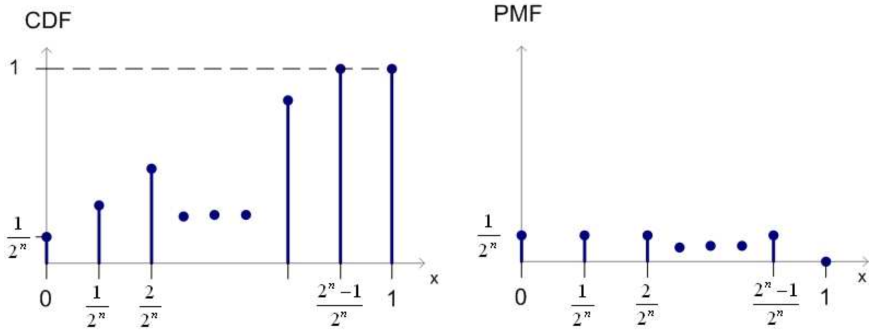

Figure 1: CDF and PMF for *std_unif_disc*

Standard Uniform random variable and the second component is the unused portion of the infinite Boolean sequence. The following properties for the discrete Standard Uniform random variable may be proved by induction on its argument n

Lemma 1:
$\vdash \forall$ n,s. $0 \leq$ fst(std_unif_disc n s) $\leq 1 - (\frac{1}{2})^n$

Lemma 2:
$\vdash \forall$ m,n,x. $\mathbb{P}\{$s | fst(std_unif_disc n s) $\leq$ x$\}$ =
    if (x < 0) then 0 else
    if (x $\geq$ 1) then 1 else
    if (x = $\frac{m}{2^n}$) then $\frac{SUCm}{2^n}$ else 0

Lemma 3:
$\vdash \forall$ m,n,x. $\mathbb{P}\{$s | fst(std_unif_disc n s) = x$\}$ =
    if (x < 0) then 0 else
    if (x $\geq$ 1) then 0 else
    if (x = $\frac{m}{2^n}$) then $\frac{1}{2^n}$ else 0

The term $\frac{m}{2^n}$ in Lemmas 2 and 3 represents the dyadic rationals since the variable m belongs to the HOL datatype *num* for natural numbers $\{0, 1, 2, \cdots\}$. Collectively Lemmas 1, 2 and 3, illustrated in Figure 1, formally prove that the first component of the function *std_unif_disc* is a discrete uniform random variable.

The function *std_unif_disc* can now be used to model the real sequence of Equation (3). We proved in HOL that this sequence is convergent, i.e., it approaches a unique value when n tends to infinity.

Lemma 4:
$\vdash \forall$ s. convergent (λn. fst(std_unif_disc n s))

where, *convergent* represents the HOL function of a convergent real sequence [13]. Based on Lemma 4, we are able to formally specify the Standard Uniform random variable in HOL according to Equation (4).

$\vdash \forall$ s. std_unif_cont s = lim (λn. fst(std_unif_disc n s)) where *lim* is the HOL function for the limit of a real sequence [13]. The following properties may be proved using the real analysis theorems [13] and the function definition for *std_unif_disc*.

```
Lemma 5:
⊢ ∀ s.  0 ≤ std_unif_cont s ≤ 1

Lemma 6:
⊢ ∀ s,n.  fst(std_unif_disc n s) ≤
                    std_unif_cont s ≤
                            fst(std_unif_disc n s)+(1/2)^n
```

Lemma 5 formally shows that the value for the function *std_unif_cont* always lies in the real interval [0,1]. The minimum and maximum values of 0 and 1 correspond to the cases when all the elements of the infinite Boolean sequence *s* are False or True respectively. Lemma 6 highlights the relationship between the values of the first component of the function *std_unif_disc* and the function *std_unif_cont*. Another equivalent form of this result is that if the value for the former is a, then the value of the later lies in the interval $[a, a + (\frac{1}{2})^n]$.

$$\begin{aligned}(fst(std\_unif\_disc\ n\ s) = a) \Rightarrow \\ (a \leq (std\_unif\_cont\ s) \leq a + (\frac{1}{2})^n)\end{aligned} \tag{5}$$

4 Verification of Standard Uniform Random Variable in HOL

In this section, we verify the correctness of the proposed specification of the Standard Uniform random variable using its probability distribution properties.

The CDF of a random variable represents the probability that its value is less than or equal to some real number x and is a distinguishing characteristic for all random variables. We formally verify in HOL that the function *std_unif_cont* correctly models the Standard Uniform random variable by proving its CDF to be equal to the theoretical CDF of a Standard Uniform random variable.

$$\mathbb{P}\{s \mid std\_unif\_cont\ s\ \leq\ x\} = \begin{cases} 0 & \text{if } x < 0; \\ x & \text{if } 0 \leq x < 1; \\ 1 & \text{if } 1 \leq x. \end{cases} \tag{6}$$

The proof for the cases (x<0) and (1≤x) is a straightforward implication of Lemma 5 which states that for all infinite Boolean sequences the value of the function *std_unif_cont* lies in the interval [0,1]. The probability that the function *std_unif_cont* acquires a value less than 0 is 0 as there is no infinite Boolean sequence that satisfies this condition. Similarly, the probability that the function *std_unif_cont* acquires a value less than or equal to 1 is 1 since all infinite Boolean sequences fulfill this condition.

```
Lemma 7:
⊢ ∀ x.  (x < 0) ⇒ ℙ{s | fst(std_unif_cont s) ≤ x} = 0
```

Lemma 8:
$\vdash \forall$ x. $(1 \leq x) \Rightarrow \mathbb{P}\{$s | fst(std_unif_cont s) $\leq$ x$\}$ = 1

Evaluating the probability of Equation (6) for the interval [0,1) is a surprisingly difficult problem in the HOL theorem prover. However, given that we have evaluated the CDF for the first component of the function *std_unif_disc*, which represents a discrete Standard Uniform random variable, a reasonable approach is to find a discrete approximation to the CDF of the function *std_unif_cont*, which represents the Standard Uniform random variable. The key to this approach is to be able to express the CDF of the function *std_unif_cont* in terms of the CDF of the first component of the function *std_unif_disc*. In order to do this, we need to identify the closest values of the first component of the function *std_unif_disc* corresponding to any given value of the function *std_unif_cont*. We known from Lemmas 1, 2 and 3 that the first component of the function *std_unif_disc* generates dyadic rationals with denominator 2^n in the interval $[0, 1-(\frac{1}{2})^n]$. On the other hand, the function *std_unif_cont* can attain any real value in the interval [0,1]. Therefore, the closest values of the first component of the function *std_unif_disc* to any given value, say y, of the function *std_unif_cont* are the two consecutive dyadic rationals (with denominators 2^n) such that the smaller dyadic rational is less than y and the greater dyadic rational is greater than or equal to y. The mathematical concept of *ceiling*, that represents the smallest integer number greater than or equal to a real number, plays a significant role in identifying these dyadic rationals. We proved in HOL that for any positive real number y the above mentioned dyadic rationals are $\frac{\lceil 2^n y \rceil - 1}{2^n}$ and $\frac{\lceil 2^n y \rceil}{2^n}$

Lemma 9:
$\vdash \forall$ n,y. $(0 \leq y) \Rightarrow \frac{\lceil 2^n y \rceil - 1}{2^n} < y \leq \frac{\lceil 2^n y \rceil}{2^n}$

where, $\lceil z \rceil$ denotes our HOL definition for the ceiling function that returns the smallest *num* value greater than or equal to its real argument z.

Now we will show how to express the CDF of the function *std_unif_cont* in terms of the CDF of the first component of the function *std_unif_disc* using the above dyadic rationals. It is important to note that the set $\{s \mid fst(std\_unif\_disc\ n\ s) \leq \frac{m}{2^n}\}$ contains all the infinite Boolean sequences for which the value of the first n bits based on the algorithm implemented by the function *std_unif_disc* is less than or equal to the dyadic rational $\frac{m}{2^n}$. Using Equation (5), we can say that the value produced by the algorithm implemented by the function *std_unif_cont*, for any infinite Boolean sequence that is present in the set $\{s \mid std\_unif\_disc\ n\ s \leq \frac{m}{2^n}\}$, must be less than or equal to $\frac{m+1}{2^n}$. We used this useful reasoning to prove the following

Lemma 10:
$\vdash \forall$ x,n. $(0 \leq x) \Rightarrow$
$\{$s | fst(std_unif_disc n s) $\leq \frac{\lceil 2^n x \rceil - 2}{2^n}\} \subseteq$
$\{$s | std_unif_cont s $\leq$ x$\} \subseteq$
$\{$s | fst(std_unif_disc n s) $\leq \frac{\lceil 2^n x \rceil}{2^n}\}$

The first set $\{s \mid fst(std\_unif\_disc\ n\ s) \leq \frac{\lceil 2^n x \rceil - 2}{2^n}\}$, based on the above reasoning, contains all the infinite Boolean sequences for which the value produced by the algorithm implemented by the function *std_unif_cont* lies in the interval $[0, \frac{\lceil 2^n x \rceil - 1}{2^n}]$. This set is a subset of the set $\{s \mid \mathit{std\_unif\_cont}\ s\ \leq\ x\}$, which contains all the infinite Boolean sequences for which the value produced by the algorithm implemented by the function *std_unif_cont* lies in the interval $[0, x]$, as $\frac{\lceil 2^n x\rceil -1}{2^n}$ is always less than x according to Lemma 9. Similarly, the set $\{s \mid \mathit{std\_unif\_cont}\ s\ \leq\ x\}$ is a subset of the set $\{s \mid \mathit{fst}(\mathit{std\_unif\_disc}\ n\ s)\ \leq\ \frac{\lceil 2^n x\rceil}{2^n}\}$, which contains all the infinite Boolean sequences for which the value produced by the algorithm implemented by the function *std_unif_cont* lies in the interval $[0, \frac{\lceil 2^n x\rceil+1}{2^n}]$, as x is always less than or equal to $\frac{\lceil 2^n x\rceil+1}{2^n}$ according to Lemma 9.

Lemma 10 and the probability axiom, which states that the probability of a measurable set is always less than or equal to the probability of its measurable superset

$$\vdash \forall\ s\ t.\ \mathit{measurable}(s) \wedge\ \mathit{measurable}(t) \wedge s \subseteq t\ \Rightarrow\ \mathbb{P}\ s \leq \mathbb{P}\ t \qquad (7)$$

can be used to obtain the required relationship between the CDFs of the function *std_unif_cont* and the first component of the function *std_unif_disc*. But, in order to use Equation (7), we must prove all the sets in Lemma 10 to be measurable, i.e., they are in $\mathcal{E}$. It has been shown in [15], that if a function accesses the infinite boolean sequence using only the `unit`, `bind` and `sdest` primitives then the function is guaranteed to be measurable and thus leads to measurable sets. The function *std_unif_disc* satisfies this condition and thus Hurd's formalization framework can be used to prove

```
Lemma 11:
⊢ ∀ x,n.  measurable {s | fst (std_unif_disc n s) ≤ x}
```

On the other hand, the definition of the function *std_unif_cont* involves the *lim* function and thus the corresponding sets can not be proved to be measurable in a very straight forward manner. Therefore, in order to prove this, we leveraged the fact that each set in the sequence $(\lambda n.\{s \mid \mathit{fst}(\mathit{std\_unif\_disc}\ n\ s)$
$\leq\ x\})$ is a subset of the set before it. Thus, the set $\{s \mid \mathit{std\_unif\_cont}\ s\ \leq\ x\}$ is equal to the countable intersection of all sets in the above sequence

```
Lemma 12:
⊢ ∀ x.  {s | std_unif_cont s ≤ x} =
              ⋂n (λ n.  {s | fst (std_unif_disc n s) ≤ x})
```

Now the set $\{s \mid \mathit{std\_unif\_cont}\ s\ \leq\ x\}$ can be proved to be measurable since measurable sets are closed under countable intersections [15] and all the sets in the sequence $(\lambda n.\{s \mid \mathit{fst}(\mathit{std\_unif\_disc}\ n\ s)\ \leq\ x\})$ are measurable according to Lemma 11.

```
Lemma 13:
⊢ ∀ x.  measurable {s | std_unif_cont s ≤ x}
```

Lemmas 10, 11 and 13 along with Equation (7) can be used to obtain the desired relationship between the CDFs. The result can be further simplified by using the CDF relation for the first component of the function *std_unif_disc* given in Lemma 2.

Lemma 14:
$\vdash \forall$ x,n. $(0 \leq x) \wedge (x < 1) \Rightarrow$
$\frac{\lceil 2^n x\rceil -1}{2^n} \leq \mathbb{P}\{\texttt{s} \mid \texttt{std\_unif\_cont s} \leq \texttt{x}\} \leq \frac{\lceil 2^n x\rceil +1}{2^n}$

As n approaches infinity both the fractions in Lemma 11 approach x. This fact led us to prove the CDF relation of Equation (6) for the interval [0,1) in the HOL theorem prover. Now, Lemmas 7, 8 and 14 can be used to prove the desired CDF property for the function *std_unif_cont*

```
Theorem 1:
⊢ ∀ x.  ℙ{s | std_unif_cont s ≤ x} =
        if (x < 0) then 0 else (if (x < 1) then x else 1)
```

Theorem 1 proves that the CDF of the function *std_unif_cont* is the same as the theoretical value of the CDF for a Standard Uniform random variable given in Equation (1) and thus is a formal argument to support the correctness of the fact that the function *std_unif_cont* models a Standard Uniform random variable.

Using similar reasoning as above, we also proved in the HOL theorem prover that the PMF of the function *std_unif_cont* is equal to 0.

```
Theorem 2:
⊢ ∀ x.  ℙ{s | std_unif_cont s = x} = 0
```

It follows from Theorem 2 that every outcome of the function *std_unif_cont* has a probability 0; which is a unique characteristic of all continuous random variables. Thus, Theorem 2 can be used to formally regard the function *std_unif_cont* as a continuous random variable.

5 Formalization of Continuous Uniform Distribution

The process of obtaining random variates with arbitrary distributions using a uniform RNG is termed as Non-uniform random number generation. All computer based RNGs generate uniformly distributed numbers [17] and nonuniform random generation methods are quite commonly used in applications which call for other kinds of distributions. Random number generation has intrigued scientists for a few decades, and a lot of effort has been spent in order to obtain efficient and accurate algorithms for various continuous random variables. The Inverse Transform method [8] is the most commonly used Non-uniform random generation method and is used to generate random variates for any continuous probability distribution for which the the inverse of the CDF can be expressed in a closed mathematical form. For generating random variates for other probability distributions that do not have a closed form expression for the CDF such as Normal, t(Student's), and Chi-Square distributions [27], either numerical approximations [4] or other specialized and complex techniques often designed for specific distributions [8] are used. For example, Knuth [17] provides different algorithms, along with their proof of correctness, for generating Normal random numbers. Recently, there has been an effort in developing so-called automatic or black-box algorithms for generating variants from an arbitrary (known) density, based on acceptance/rejection with a transformed density [14].

We believe that our formalized Standard Uniform random variable can be used in conjunction with the enormous amount of research available in the field of Non-uniform random number generation to formalize other continuous probability distributions in a higher-order logic theorem prover. In this section, we illustrate this by formalizing the Continuous Uniform random variable in the HOL theorem prover.

Continuous Uniform random variable is a random variable for which the probability that it will belong to a subinterval of [a,b] is proportional to the length of that subinterval. It can be characterized by the CDF as follows

$$\mathbb{P}(X \leq x) = \begin{cases} 0 & \text{if } x \leq a; \\ \frac{x-a}{b-a} & \text{if } a < x \leq b; \\ 1 & \text{if } b < x. \end{cases} \tag{8}$$

The Continuous Uniform random variable can be formalized using our formalized Standard Uniform random variable and the Inverse Transform method [8] which is based on the following proposition:

> *Let U be a Standard Uniform random variable. For any continuous distribution F, the random variable X defined by $X = F^{-1}(U)$ has distribution F, where $F^{-1}(u)$ is defined to be that value of x such that $F(x) = u$.*

A formal proof for the above proposition may be found in [8]. The Continuous Uniform [a,b] random variable can be formally specified using the Inverse Transform method as follows

```
⊢ ∀ a,b,s.
      uniform_cont a b s = (b - a) * (std_unif_cont s) + a
```

The probabilistic function $uniform\_cont$ accepts two real valued parameters a, b and the infinite Boolean sequence s and returns a real number in the interval [a,b]. It can be verified that the function $uniform\_cont$ correctly models a Continuous Uniform random variable by proving its CDF ($\mathbb{P}\{s|uniform\_cont\ a\ b\ s\ \leq\ x\}$) to be equal to the theoretical value given in Equation (8). We first used the definition of the function $uniform\_cont$ and elementary real arithmetic operations to transform the set $\{s|uniform\_cont\ a\ b\ s$ $\leq\ x\}$ in such a way that ($std\_unif\_cont\ s$) is the only term that remains on the left side of the inequality ($\mathbb{P}\{s|std\_unif\_cont\ s \leq \frac{x-a}{b-a}\}$). Now, the PMF and CDF properties for the function std_unif_cont along with some simple arithmetic reasoning can be used to prove the CDF relation for $uniform\_cont$ in HOL.

```
Theorem 3:
⊢ ∀ a,b,x.  (a < b) ⇒
      ℙ{s | uniform_cont a b s ≤ x} =
        if (x ≤ a) then 0 else (if (x ≤ b) then (x−a)/(b−a) else 1)
```

Similarly, our Standard Uniform random variable can be used to formalize other continuous distributions, such as $Triangular(0, a)$, $Exponential(\lambda)$, $Pareto(a, b)$ and $Rayleigh(\sigma)$,using the Inverse Transform method [8].

6 Probabilistic Analysis of Roundoff Error

The formalization of the continuous random variables allows us to formally express the continuous randomness found in many scientific and engineering systems. This leads to the evaluation of the related probabilistic quantities within the environment of the HOL theorem prover. Due to the soundness of the theorem prover, the probabilistic analysis conducted

Simulation Samples (N)	Count (C)	$\mathbb{P}(-4\text{x}10^{-12} \leq X \leq 4\text{x}10^{-12})$	Error
100	92	0.9200	$+0.1200$
	65	0.6500	-0.1500
1000	767	0.7670	-0.0330
	785	0.7850	-0.0150
10000	7953	0.7953	-0.0047
	8087	0.8087	$+0.0087$

Table 2: Simulation Results

in this way is without any approximation errors. In this section, we present the practical effectiveness of the proposed approach by comparing its results with a commonly used Monte Carlo simulation. For illustration purposes, we have chosen a simplified probabilistic analysis example of roundoff error in a digital processor.

In order to optimize the performance of a digital processor, an engineering team is interested to know the probability of the event when the roundoff error fluctuates the actual value by $\pm 4\text{x}10^{-12}$. If the probability of this event is less than 0.7 then the engineers think that more precision is required, i.e., the word length of the processor needs to be increased. It is quite well known, from statistical analysis, that the Continuous Uniform distribution is used to model roundoff errors [28]. In this particular example, the roundoff error, X, is uniformly distributed over the interval [-5x10^{-12}, 5x10^{-12}]. The probability under question can be found theoretically by integrating over the *probability density function* (PDF) of the Continuous Uniform distribution as follows:

$$\mathbb{P}(-4\text{x}10^{-12} \leq X \leq 4\text{x}10^{-12}) = \int_{-4\text{x}10^{-12}}^{4\text{x}10^{-12}} \frac{1}{5\text{x}10^{-12} - (-5\text{x}10^{-12})} dt = \frac{8}{10} = 0.8 \quad (9)$$

The Monte Carlo method may be applied to approximately obtain the above probability by generating a specific number, say N, of uniformly distributed random numbers in the range [-5x10^{-12}, 5x10^{-12}] and keep track of all the numbers that have a value greater than -4x10^{-12} or less than 4x10^{-12} in a variable, say C. The desired probability can be obtained by dividing the value of C by N. We implemented this algorithm in MATLAB and ran simulations for different values of N. The results are presented in Table 2.

The first observation from the above exercise is that the simulation based approach returned an approximate result, as expected. Secondly, it can be easily observed that the accuracy of the results is dependant on the value of N; the more random numbers we generate the closer the results get to the theoretical value of Equation (9). On the other hand, an increase in the value of N increases the number of random number generations and the simulation cycles which in turn means an increase in the CPU time. Also note that, one of the simulation results returned a probability of 0.65 which could mislead the engineers to change the word length.

The formalized Continuous Uniform random variable of Section 5, represented by the function $uniform\_cont$, can be used to perform the same probabilistic analysis in the environment of the HOL theorem prover. The first step is to express the desired probability in terms of CDF relations. This can be done by using the following probability axiom

$$\vdash \forall\ s\ t.\ \mathit{measurable}(s)\ \wedge\ \mathit{measurable}(t) \wedge (s \cap t = \{\})\ \Rightarrow \\ (\mathbb{P}\ (s \cup t) = \mathbb{P}\ s + \mathbb{P}\ t) \tag{10}$$

Lemma 13 can be used to prove that the sets of the form $\{s|\mathit{uniform\_cont}\ a\ b\ s \leq\ x\}$ are measurable. Thus, using Equation (10) we can prove

```
Lemma 15:
⊢ ∀ a,b,x,y.  (a < b) ⇒
       ℙ{s | (x ≤ uniform_cont a b s) ∧
                  (uniform_cont a b s ≤ y)} =
                          ℙ{s | uniform_cont a b s ≤ y} -
                          ℙ{s | uniform_cont a b s ≤ x}
```

Now, the CDF relation for the function $\mathit{uniform\_cont}$, that we proved in Theorem 3, can be utilized to evaluate the probability under consideration by specializing lemma 15 for the case when a, b, x and y are equal to -5x10^{-12}, 5x10^{-12}, -4x10^{-12} and 4x10^{-12} respectively. This leads us to obtain the exact same result of Equation (9) in the HOL theorem prover

$$\vdash \mathbb{P}\{\texttt{s | (-4x10}^{-12} \leq \texttt{uniform\_cont -5x10}^{-12}\ \texttt{5x10}^{-12}\ \texttt{s)} \wedge \\ \texttt{(uniform\_cont -5x10}^{-12}\ \texttt{5x10}^{-12}\ \texttt{s} \leq \texttt{4x10}^{-12}\texttt{)}\} \texttt{ = 0.8}$$

This simple example of probabilistic analysis supports our claim of achieving 100% accuracy using the proposed approach. Another major advantage of using interactive theorem proving for probabilistic analysis is the generality of the approach, i.e., Lemma 15 can be used to obtain the desired probability for any values of a, b, x and y, Whereas in the case of simulation, we have to perform the analysis all over again if the need arises to find the probability for some different parameters. This characteristic of interactive theorem proving can be very useful for tweaking performance or reliability parameters.

This simple example, illustrates the accuracy related shortcomings of the simulation based techniques and these issues get worse as the analysis become more complex. On the other hand, the proposed theorem proving approach is capable of providing precise reasoning irrespective of the problem's complexity. It is important to note here that the proposed theorem proving based approach cannot be regarded as the golden solution in performing probabilistic analysis because of its own limitations. Dealing with analytically complex problems with theorem-provers usually requires a proficient user and a significant amount of formalization. Where as the simulation based approach offers automated, push-button type, solutions irrespective of the analytical complexity of the problem. Therefore, we consider simulation and higher-order theorem proving as complementary techniques, i.e., the methods have to play together for a successful probabilistic analysis framework. The proposed theorem proving based approach can be used for the safety critical parts of the design and simulation based approaches can handle the rest.

7 Related Work

Due to the vast application domain of continuous random variables, many researchers around the world are trying to improve the modeling techniques for continuous probability distributions in computer based environments. The ultimate goal is to come up with a probabilistic analysis framework that includes robust and accurate analysis methods, has the ability to perform analysis for large-scale problems and is easy to use. In this section, we provide a brief account of the state-of-the-art and some related work in this field.

Hurd's PhD thesis, reviewed in Section 2.2 of this paper, presents a methodology to formalize probabilistic algorithms in higher-order-logic. Hurd's main contributions include the formalization of the mathematical measure theory and probability theory along with the development of a framework to formalize probabilistic algorithms in the HOL theorem prover. Hurd also presented the formalization of several discrete random variables but did not touch the topic of formalizing continuous random variables, which has been the main topic of discussion in the current paper. We have proposed to build upon Hurd's PhD thesis to formalize the continuous random variables so that the HOL theorem prover can be used as a probabilistic analysis platform for systems which include continuous random quantities.

A number of *probabilistic languages*, e.g., `Probabilistic cc` [12], λ_o [23] and `IBAL` [25], have been proposed that are capable of modeling random variables. Probabilistic languages treat probability distributions as primitive data types and abstract from their representation schemes. Therefore, they allow programmers to perform probabilistic computations at the level of probability distributions rather than representation schemes. These probabilistic languages are quite expressive and have been shown to express most continuous probability distributions but they have their own limitations. For example, either they require a special treatment such as the lazy list evaluation strategy in `IBAL` and the limiting process in `Probabilistic cc` or they do not support precise reasoning as in the case of λ_o. The proposed theorem proving based approach, on the other hand, is not only capable of formally expressing most continuous probability distributions but also to precisely reason about them.

It is interesting to note that one of the probabilistic languages, λ_o, proposed by Park *et. al* in [23] is based on sampling functions. A sampling function is defined as a mapping from the unit interval [0,1] to a probability domain $\mathfrak{D}$. Given a random number drawn from a Standard Uniform distribution, it returns a sample in $\mathfrak{D}$, and thus specifies a unique probability distribution. Thus, this approach is very similar to what we have proposed in this paper, as it also utilizes the Standard Uniform random variable to obtain other continuous random variables. [23] contains sampling algorithms for various continuous random variables which can be utilized to formalize the respective random variables in the HOL theorem prover using our formalized Standard Uniform random variable.

Another alternative for formal probabilistic verification is to use probabilistic model checking techniques, e.g., [1], [2], [7], [26]. It involves the construction of a precise mathematical model of the probabilistic system to be analyzed, formal specification of properties of this system and then analysis of these properties based on an exhaustive exploration of the constructed model. This approach is capable of providing precise solutions in an automated way; however it is limited for systems that can only be expressed as a probabilistic finite state machine. Our proposed theorem proving based approach, in contrast, is capable of handling all kinds of probabilistic systems including the *unbounded* ones, as demonstrated by the example in Section 6. Another major limitation of the probabilistic model checking approach is the state space explosion [6], which is not an issue with our approach.

Knuth and Yao [18] presented procedures that minimize the average number of bits required to generate random numbers from various continuous probability distributions in arbitrary systems of notation. Most of the sampling algorithms that have been analyzed in this paper are based on the Standard Uniform random variable. An interesting topic for further research is to utilize our formalized Standard Uniform random variable to verify Knuth and Yao's results in the HOL theorem prover.

8 Conclusions

We have presented a formalization methodology for the Standard Uniform random variable in higher-order logic. This formalization is verified to be correct in the HOL theorem prover. We also demonstrated that the formalized Standard Uniform random variable can be used in conjunction with Non-uniform random number generation methods to formalize other continuous random variables. To the best of our knowledge, this is the first time that a methodology for the formal reasoning of continuous probability distributions in a mechanical theorem prover has been proposed. Therefore, the presented work can be considered a significant step towards the development of a formal reasoning framework for systems involving continuous probability distributions.

The proposed formalization opens the doors for a new era in the fields of computer science, operations research and statistics. Computer scientists may use the formalized continuous random variables in program verification and comparisons of algorithms. In operations research, the formalized probabilistic analysis may be used to complement large scale simulations. Statisticians may use the formalized continuous random variables to formally test and compare estimators before using them in real life.

As a next step towards a complete formalization framework for continuous random variables, we are formalizing the Inverse Transform method itself in HOL. This will considerably ease the formalization process for the corresponding continuous random variables. Similarly, other Non-uniform random generation methods, such as Box-Muller and acceptance/rejection methods [8] can also be formalized. In order to have a complete formal probabilistic analysis framework, it is also essential to formalize the theory of expectation and the basic statistical quantities of mean, moment, and variance.

References

[1] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic Model Checking for Probabilistic Processes. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 430–440. Springer-Verlag, 1997.

[2] C. Baier, B. Haverkort, H. Hermanns, and J. P. Katoen. Model Checking Algorithms for Continuous time Markov chains. *IEEE Transactions on Software Engineering*, 29(4):524–541, 2003.

[3] P. Billingsley. *Probability and Measure.* John Wiley, 3^{rd} edition, 1995.

[4] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation.* Springer-Verlag, 2^{nd} edition, 1987.

[5] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, 2000.

[7] C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *Journal of the ACM*, 42(4):857–907, 1995.

[8] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[9] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, 1971.

[10] M. J. C. Gordon. Mechanizing Programming Logics in Higher-0rder Logic. In *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 387–439. Springer-Verlag, 1989.

[11] M. J. C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.

[12] V. T. Gupta, R. Jagadeesan, and P. Panangaden. Stochastic Processes as Concurrent Constraint Programs. In *Principles of Programming Languages*, pages 189–202. ACM Press, 1999.

[13] J. Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998.

[14] W. Hörmann and J. Leydold. Automatic Random Variate Generation for Simulation Input. In *Winter Simulation Conference*, pages 675–682. IEEE Press, 2000.

[15] J. Hurd. *Formal Verification of Probabilistic Algorithms.* PhD thesis, University of Cambridge, Cambridge, UK, 2002.

[16] W. J. Kennedy and J. E. Gentle. *Statistical Computing.* Marcel-Dekker, 1980.

[17] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley Professional, 1998.

[18] D. E. Knuth and A. C. Yao. The Complexity of Nonuniform Random Number Generation. *Algorithms and Complexity: New Directions and Recent Results*, pages 357–428, 1976.

[19] D. J. C. MacKay. Introduction to Monte Carlo methods. In *Learning in Graphical Models, NATO Science Series*, pages 175–204. Kluwer Academic Press, 1998.

[20] B. D. McCullough. Assessing the Reliability of Statistical Software: Part I. *The American Statistician*, 52(4):358–366, 1998.

[21] B. D. McCullough. Assessing the Reliability of Statistical Software: Part II. *The American Statistician*, 53(2):149–159, 1999.

[22] R. Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[23] S. Park, F. Pfenning, and S. Thrun. A Probabilistic Language based upon Sampling Functions. In *Principles of Programming Languages*, pages 171–182. ACM Press, 2005.

[24] L. C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 2^{nd} edition, 1996.

[25] A. Pfeffer. IBAL: A Probabilistic Rational Programming Language. In *International Joint Conferences on Artificial Intelligence*, pages 733–740. Morgan Kaufmann Publishers, 2001.

[26] J. Rutten, M. Kwaiatkowska, G. Normal, and D. Parker. Mathematical Techniques for Analyzing Concurrent and Probabilisitc Systems. *CRM Monograph*, 23, 2004.

[27] K. S. Tridevi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* Wiley-Interscience, 2002.

[28] B. Widrow. Statistical Analysis of Amplitude-quatized Samled Data Systems. *AIEE Trans. (Applications and Industry)*, 81:555–568, January 1961.