Hasan O, Tahar S. Formally analyzing expected time complexity of algorithms using theorem proving. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(6): 1305–1320 Nov. 2010. DOI 10.1007/s11390-010-1103-6

Formally Analyzing Expected Time Complexity of Algorithms Using Theorem Proving

Osman Hasan and Sofiène Tahar, *Senior Member, IEEE, Member, ACM,*

Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada

E-mail: {o_hasan, tahar}@ece.concordia.ca

Received June 20, 2009; revised July 26, 2010.

Abstract Probabilistic techniques are widely used in the analysis of algorithms to estimate the computational complexity of algorithms or a computational problem. Traditionally, such analyses are performed using paper-and-pencil proofs and the results are sometimes validated using simulation techniques. These techniques are informal and thus may result in an inaccurate analysis. In this paper, we propose a formal technique for analyzing the expected time complexity of algorithms using higher-order-logic theorem proving. The approach calls for mathematically modeling the algorithm along with its inputs, using indicator random variables, in higher-order logic. This model is then used to formally reason about the expected time complexity of the underlying algorithm in a theorem prover. The paper includes the higher-order-logic formalization of indicator random variables, which are fundamental to the proposed infrastructure. In order to illustrate the practical effectiveness and utilization of the proposed infrastructure, the paper also includes the analysis of algorithms for three well-known problems, i.e., the hat-check problem, the birthday paradox and the hiring problem.

Keywords formal method, higher-order logic, probability theory, theorem proving, birthday paradox, hat-check problem, hiring problem

1 Introduction

An algorithm, which may be defined as a sequence of computational steps that transforms the given input parameters into the desired output, is the most fundamental component of computer programming. The computational complexity of the underlying algorithms greatly affects the overall efficiency of virtually all applications of computer science, ranging from combinatorial optimization, machine learning, data streaming, complexity theory, coding theory, to communication networks and secured protocols. Thus, a significant amount of time and effort is spent on analyzing several candidate algorithms for one problem in order to identify the most efficient solution[1]. For example, various sorting algorithms can be analyzed to find the fastest one for sorting n numbers. The biggest challenge in such analysis is the fact that the inputs to the algorithms usually arrive in a random or unpredictable fashion and thus cannot be modeled in a straightforward manner for analysis purposes. One pessimistic solution to this problem is to analyze the algorithm under the worst possible scenarios. However, it is an old observation in quite a few application areas that the worst-case input patterns are not typical and might never occur in practice. So worst-case analysis can improperly suggest that the performance of the algorithm is poor. Probabilistic techniques are thus utilized in this endeavor. The main idea behind the probabilistic approach is to model the input behavior of the given algorithm by an appropriate random variable and utilize this information to judge the average or expected value of the algorithm's computational runtime[2].

The probabilistic analysis of algorithms and the usage of expectations to evaluate their complexities are widely used concepts since their introduction about a few decades ago[3-4]. The three mainstream approaches for conducting such analysis are paper-and-pencil proof methods (e.g., [5]), computer simulations (e.g., [6]), and computer algebra systems (e.g., [7]). Due to the complex nature of the present age algorithms, the traditional paper-and-pencil based proof techniques always have some risk of an erroneous analysis due to the human-error factor. Most simulation or testing based algorithm analysis softwares provide a programming environment for defining functions that approximate random variables for probability distributions. The randomness and the input patterns in algorithms are modeled by these functions and the system is analyzed using computer simulation techniques[8], such as the

Regular Paper

Monte Carlo Method[9] where the main idea is to approximately answer a query on a probability distribution by analyzing a large number of samples. Statistical quantities, such as expectation and variance, may then be calculated, based on the data collected during the sampling process, using their mathematical relations in a computer. Due to the inherent nature of simulation coupled with the usage of computer arithmetic, the analysis results attained by the simulation approach can never be termed as 100% accurate. Similarly, computer algebra systems, such as Maple, Mathematica etc., have also been used for the complexity analysis of computational algorithms[10]. Even though, computer algebra systems yield high precision numeric results by using exact fractions, arbitrary precision integers, and variable precision floating point numbers, they also fail to guarantee 100% precision of results. The main reasons being the usage of computer arithmetic systems, such as floating or fixed point representations, in computations involving real numbers and the fact that computer algebra system are constructed using extremely complicated algorithms, which are quite likely to contain bugs. For example, the work reported in [11] clearly highlights the inaccuracy limitations of a computer algebra system, using Maple as an example.

Formal methods are capable of conducting precise system analysis and thus overcome the limitations of the above mentioned traditional approaches[12]. The main principle behind formal analysis of a system is to construct a computer based mathematical model of the given system and formally verify, within a computer, that this model meets rigorous specifications of intended behavior. Two of the most commonly used formal verification methods are model checking[13] and higher-order-logic theorem proving[14]. Model checking is an automatic verification approach for systems that can be expressed as a finite-state machine. Higher-order-logic theorem proving, on the other hand, is an interactive verification approach that allows us to mathematically reason about system properties by representing the behavior of a system in higher-order logic.

The precision and accuracy of algorithm complexity analysis has become imperative these days because of their extensive usage in safety and financial critical areas, such as medicine, transportation and stock exchange markets. Therefore, more reliable analysis techniques, like formal methods, are required. In fact, they have already been used for this purpose. For example, building upon the measure theoretic formalization of probability theory, Hurd[15] presented an approach to formalize probabilistic algorithms and formally reason about their probability distribution properties using a higher-order-logic theorem prover. A very comprehensive account of existing methods for formal reasoning about probabilistic algorithms is presented in [16]. The probabilistic guarded-command language (pGCL), which is used to describe probabilistic programs in [16], has also been formalized in higher-order-logic in [17]. This formalization facilitates formal analysis of distributed random algorithms in higher-order logic. All these above mentioned existing works have been mainly targeted towards the formal specification of algorithms with probabilistic components and the ability to formally reason about their probability distribution properties. Though, to the best of our knowledge, there is no existing work that explicitly deals with the formal analysis of expected time complexity of an algorithm, which is the main focus of this paper.

The proposed approach for the formal expected time complexity analysis is based on higher-order-logic theorem proving. Higher-order logic is a system of deduction with a precise semantics and can be used for the precise specification of almost all classical mathematics theories and software systems. Interactive theorem proving is the field of computer science and mathematical logic concerned with precise computer based formal proof tools that require some sort of human assistance. The foremost criteria for the development of a higher-order-logic theorem proving based expected time complexity analysis framework are (i) to be able to model the algorithms that need to be analyzed in higher-order logic, and (ii) to be able to formally express and verify expectation properties regarding the computational runtimes of the given algorithms in a theorem prover. We propose to model the algorithms in terms if *indicator random variables*[18], which in turn can be formalized based on the approach given in [15]. Basically, an indicator random variable is a random variable with only two possible outcomes, i.e., 0 or 1. The name indicator random variable is used because the value 1 is often used to indicate the presence of an event. Indicator variables are found to be quite useful for representing situations in which we perform repeated random trials, and thus are very frequently used to model algorithms in their probabilistic analysis. In order to facilitate this step, the paper provides the higher-order logic model for an indicator random variable and the verification of some of its key properties. For the second step, which is expressing and reasoning about the expectation of computational runtime of algorithms, we propose to use the higher-order-logic model of the algorithm, developed in the first step, along with the higher-order-logic formalization of expectation, given in [19].

In order to illustrate the utilization and effectiveness of the proposed higher-order-logic theorem proving based framework for handling real-world algorithm analysis problems, we analyze the expected time complexity of three commercially used algorithms, i.e., the hat-check problem[20], the birthday paradox[21] and the hiring problem[18]. The *hat-check problem* is a classic combinatorial question, sometimes also referred to as the Montmort's matching problem (since one of its variants was first proposed by mathematician de Montmort in his 1708 treatise on the analysis of games of chance[22]). The problem is about finding the right hat for a group of men that have checked their hats in a restaurant where the tickets got scrambled somehow. Because of its wide range of applications the problem has been studied by many mathematicians (e.g., [23-25]). The *birthday paradox*, or birthday problem, refers to the probability that in a set of randomly chosen people some pair of them will have the same birthday. It is a widely used characteristic in congruity[26], combinatorics[27-28] and computer security literature[29]. Whereas the *hiring problem*, sometimes also referred to as the classical secretary problem, highlights the problem of choosing the best of a set of randomly presented candidates. The hiring problem captures fundamental issues and inevitable tradeoffs related to making irrevocable decisions under an uncertain future. Its application spans multiple scientific disciplines, such as mathematics, economics and computer science, and thus since its introduction in the 1960's[30], the hiring problem has been the subject of many papers (e.g., [31-34]). In this paper, we present the higher-order-logic formalization of algorithms for all of the above mentioned three problems using the proposed indicator random variables based approach and the details about their expected time complexity analysis of using a theorem prover. The analysis results have been found to be 100% precise, which to the best of our knowledge is an achievement that has not been reported for these or any other similar algorithms in the open literature so far. Also, it is important to note here that the proposed approach is not limited to the algorithms of the above three problems and instead is generic enough to formally analyze many other algorithms. We have chosen the above three mainly because we believe that they cover many interesting and distinct probabilistic analysis cases as will be discussed later in the paper.

The proposed work is done using the HOL theorem prover[35], which is based on higher-order logic. The main motivation behind this choice is the fact that most of the work that we build upon is developed in HOL. It is important to note here that the ideas presented in this paper are not specific to the HOL theorem prover and can be adapted to any other higher-order-logic theorem prover as well, such as Isabelle[36], Coq[37] or PVS[38].

The rest of the paper is organized as follows. Section 2 provides a review of related work. Then, in Section 3, we present a brief introduction to the HOL theorem prover. Next, Section 4 highlights upon the two fundamental components that we build upon for analyzing the expected time complexity of algorithms in a higher-order-logic theorem prover, i.e., modeling random variables in higher-order logic and formally verifying their expectation properties. This is followed by the description of our higher-order-logic definition of the indicator random variable along with the formal verification of some of its key properties in Section 5. We utilize this infrastructure in Section 6 to analyze the hat-check problem, birthday paradox and hiring problem. Finally, Section 7 concludes the paper.

2 Related Work

The early foundations of probabilistic analysis in a higher-order-logic theorem prover were laid down by Nędzusiak[39] and Bialas[40] when they proposed a formalization of some measure and probability theories in higher-order logic. Hurd[15] implemented their work and developed a framework for the verification of probabilistic algorithms in the HOL theorem prover. The algorithms, along with their random components, can be formalized as higher-order-logic functions and formally verified, based on the corresponding probability distribution properties, using the methodology proposed in [15]. Random variables are fundamentally probabilistic algorithms and thus they can also be formalized based on Hurd's approach. In fact, building upon Hurd's formalization, most of the commonly used discrete[15] and continuous[41] random variables have been formalized in higher-order-logic and their corresponding probabilistic[15] and statistical[19] properties have been verified using interactive theorem proving techniques. These formalized random variables can in turn be used to express random or unpredictable phenomenon in system models and the probabilistic analysis of these system models can be conducted in a theorem prover using the corresponding probabilistic and statistical properties of these random variables. Some of the higher-order-logic theorem proving based probabilistic analysis examples include the performance analysis of real-time systems[42], communication protocols[43], wireless systems[44] and safety analysis of fabrication faults[45].

The above mentioned results have also been used for the probabilistic analysis of algorithms. For example, Hurd utilized his infrastructure to analyze the symmetric simple random walk and the Miller-Rabin primality test based on the corresponding probability distribution properties[15]. Similarly, we utilized our theories related to the formal verification of statistical properties for the performance analysis of the Coupon Collector's problem[19]. What makes the analysis presented in the current paper different from these past endeavors is the fact that it presents an indicator random variable based approach for the analysis of expected time complexity of algorithms in a higher-order-logic theorem prover, which to the best of our knowledge is a novelty that has not been reported in the open literature so far.

Besides theorem proving, probabilistic model checking is the second most widely used formal probabilistic analysis method[46-47]. Like traditional model checking[48], probabilistic model checking involves the construction of a precise state-based mathematical model of the given probabilistic system, which is then subjected to exhaustive analysis to verify if it satisfies a set of probabilistic properties formally expressed in some appropriate logic. Numerous probabilistic model checking algorithms and methodologies have been proposed in the open literature, e.g., [49-50], and based on these algorithms, a number of tools have been developed, e.g., PRISM[51] and VESTA[52]. Besides the accuracy of the results, another promising feature of probabilistic model checking is the ability to perform the analysis automatically. On the other hand, probabilistic model checking is limited to systems that can only be expressed as probabilistic finite state machines or Markov chains. Another major limitation of the probabilistic model checking approach is state space explosion[48]. Similarly, to the best of our knowledge, it has not been possible to precisely reason about statistical relations, such as expectation and variance, using probabilistic model checking so far. Probabilistic model checking has been used for the analysis of randomized distributed algorithms[53] but has been found to be incapable of conducting fully automated proofs of correctness mainly because of its limitedness to only complete and finite-state models. Higher-order-logic theorem proving, on the other hand, overcomes the limitations of probabilistic model checking and thus allows conducting formal probabilistic analysis of algorithms but at the cost of significant user interaction.

3 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[54] along with Hindley-Milner polymorphism[55] 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.

In order to ensure secure theorem proving, the logic in the HOL system is represented in the strongly-typed functional programming language ML[56]. An ML abstract data type is 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. The HOL core consists of only 5 basic axioms and 8 primitive inference rules, which are implemented as ML functions. Soundness is assured as every new theorem must be verified by applying these basic axioms and primitive inference rules or any other previously verified theorems/inference rules.

HOL supports two types of interactive proof methods: forward and backward. In forward proof, the user starts with previously proved theorems and applies inference rules to reach the desired theorem. In most cases, the forward proof method is not the easiest solution as it requires the exact details of a proof in advance. A backward or a goal directed proof method is the reverse of the forward proof method. It is based on the concept of a *tactic*; which is an ML function that breaks goals into simple subgoals. In the backward proof method, the user starts with the desired theorem or the main goal and specifies tactics to reduce it to simpler intermediate subgoals. Some of these intermediate subgoals can be discharged by matching axioms or assumptions or by applying built-in decision procedures. The above steps are repeated for the remaining intermediate goals until we are left with no further subgoals and this concludes the proof for the desired theorem.

The HOL theorem prover includes many proof assistants and automatic proof procedures[57] to assist the user in directing the proof. 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 facilitate reutilization of verified theorems, HOL allows its users to store a collection of valid HOL types, constants, axioms and theorems as an HOL theory file in computers. Once stored, HOL theories can be loaded in the HOL system and the corresponding definitions and theorems can be utilized right away. Thus, HOL theories allow us to build upon existing results in an efficient way without going through the tedious process of regenerating these results using the basic axioms and primitive inference rules. Various mathematical concepts have been formalized and saved as HOL theories by the HOL users. Out of this useful library of HOL theories, we utilized the theories of Booleans, lists, sets, positive integers, real numbers, measure and probability in this paper. In fact, one of the primary motivations of selecting the HOL theorem prover for our work was to benefit from these built-in mathematical theories.

Table 1 provides the mathematical interpretations of some frequently used HOL symbols and functions, which are inherited from existing HOL theories and will be used in the rest of the paper.

Table 1. HOL Symbols and Functions

HOL Symbols	Meaning
$\wedge$	Logical *and*
$\vee$	Logical *or*
$\neg$	Logical *negation*
`a b`	Multiplication $(a * b)$
[ ]	Empty list
::	Adds a new element to a list
++	Appends two lists together
`el n L`	n-th element of list L
`mem a L`	a is a member of list L
`length L`	Length of list L
(a, b)	A pair of two elements
`fst`	First component of a pair
`snd`	Second component of a pair
λx.t	Function that maps x to $t(x)$
{x|P(x)}	Set of all x such that $P(x)$
sum(0, k)(λn.f(n))	$\sum_{n=0}^{k-1} f(n)$
suminf(λn.f(n))	$\lim_{k\to\infty} \sum_{n=0}^{k} f(n)$
summable(λn.f(n))	$\exists x. \lim_{k\to\infty} \sum_{n=0}^{k} f(n) = x$

4 Probabilistic Analysis in HOL

The foremost criteria for conducting the expected time complexity analysis of an algorithm in a higher-order-logic theorem prover is to be able to formalize random variables in higher-order logic and verify their expectation properties. This section provides some information about these capabilities with the intent of introducing the underlying concepts along with some notations that are going to be used in the rest of the paper.

Hurd's Ph.D. Dissertation[15] can be considered a pioneering work in regards to the formalization of random variables in higher-order-logic. Random variables are fundamentally probabilistic functions that can be modeled in higher-order logic as deterministic functions with access to an infinite Boolean sequence B^{∞}; a source of infinite random bits[15]. 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 functions terminate, they return the result along with the remaining portion of the infinite Boolean sequence to be used by other programs. Thus, a random variable which takes a parameter of type α and ranges over values of type β can be represented in HOL by the function.

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

As an example, consider the Bernoulli($\frac{1}{2}$) random variable that returns 1 or 0 with equal probability $\frac{1}{2}$. It can be formalized in HOL 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 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 =
    g(fst(f s))(snd(f s)).
```

The `unit` operator is used to lift values to the monad, and the `bind` is the monadic analogue of function application. All 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
    P(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 space. 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 verify the basic laws of probability in the HOL theorem prover. For example, the additive law, which represents the probability of two disjoint events as the sum of their probabilities, can be formally verified as follows:

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

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

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

where the HOL function `fst` selects the first component of a pair and $\{x|C(x)\}$ represents a set of all x that satisfy the condition C.

The *measurability* and *independence* of a probabilistic function are important concepts in probability theory. A property `indep`, called *strong function independence*, is introduced in [15] such that if $f \in$ `indep`, then all sets involving the function f will be both measurable and independent. In this approach, a set of infinite Boolean sequences, S, is said to be measurable if and only if it is in $\mathcal{E}$, i.e., $\mathrm{S} \in \mathcal{E}$. 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 are measurable. It has been shown in [15] that a function is guaranteed to preserve *strong function independence*, if it accesses the infinite Boolean sequence using only the `unit`, `bind` and `sdest` primitives. All reasonable probabilistic programs preserve *strong function independence*, and these extra properties are a great aid to verification.

The above approach has been successfully used to formalize both discrete[15] and continuous random variables[41] and verify them based on their corresponding probability distribution properties. In this paper, we utilize the models for Bernoulli and Uniform random variables formalized as the higher-order-logic functions `ber_rv` and `unif_rv`, respectively, and verified using the following probability mass function (PMF) relations[15]:

Lemma 1 (PMF of Bernoulli(p) R.V.).

$$\vdash \forall\ \mathtt{p}.\ \ 0 \leqslant \mathtt{p} \wedge \mathtt{p} \leqslant 1 \Rightarrow \mathbb{P}\ \{\mathtt{s}\ |\ \mathtt{fst\ (ber\_rv\ p\ s) = 1}\} = \mathtt{p}.$$

Lemma 2 (PMF of Uniform(m) R.V.).

$$\vdash \forall\ \mathtt{m\ x}.\ \ \mathtt{x} < \mathtt{m} \Rightarrow \mathbb{P}\ \{\mathtt{s}\ |\ \mathtt{fst\ (unif\_rv\ m\ s) = x}\} = \frac{1}{\mathtt{m}}.$$

The function `ber_rv` for the Bernoulli(p) random variable models an experiment with two outcomes; `1` and `0`, whereas the parameter `p` represents the probability of obtaining a `1`. Whereas, the function `unif_rv` for the Uniform(m) random variable assigns equal probability to each element in the set {`0, 1,..., (m-1)`} and thus ranges over a finite number of positive integers.

Expectation theory plays a vital role in the domain of probabilistic analysis of algorithms as it is a lot easier to judge performance issues based on the average characteristic of an algorithm, which is a single number, rather than its distribution function. Building on the above mentioned probabilistic analysis infrastructure, the expectation of a discrete random variable can be defined as a higher-order-logic function as follows[19]:

Definition 1 (Expectation of Discrete R.V.).

$$\vdash \forall\ \mathtt{R}.\ \mathtt{expec\ R} = \mathtt{suminf}\ (\lambda \mathtt{n}.\mathtt{n}\mathbb{P}\ \{\mathtt{s}\ |\ \mathtt{fst\ (R\ s) = n}\})$$

where `suminf` *represents the HOL formalization of the infinite summation of a real sequence*[58] *as outlined in Table* 1. *The function* `expec` *accepts the random variable* `R` *with data type $B^\infty \to$ (positive integer $\times B^\infty$), and returns a real number. This function can be used to successfully verify the expectation relation of any discrete random variable that attains values in positive integers. For example, the higher-order-logic theorem corresponding to the expectation of the Bernoulli random variable has been formally verified in* [19] *and is given as follows.*

Lemma 3 (Expectation of Bernoulli(p) R.V.).

$$\vdash \forall\ \mathtt{p}.\ \ 0 \leqslant \mathtt{p} \wedge \mathtt{p} \leqslant 1 \Rightarrow \mathtt{expec}\ (\lambda \mathtt{s}.\ \ \mathtt{ber\_rv\ p\ s}) = \mathtt{p}$$

where $(\lambda x.t)$ *represents a lambda abstraction function in HOL that maps its argument* x *to* $t(x)$.

The linearity of expectation property, according to which the expectation of the sum of random variables equals the sum of their individual expectations,

$$Ex\Big[\sum_{i=1}^{n} R_i\Big] = \sum_{i=1}^{n} Ex[R_i] \qquad (1)$$

is one of the most important properties of expectation. It allows us to verify the expectation properties of random behaviors involving multiple random variables without going into the complex verification of their joint probability distribution properties. For facilitating the analysis of systems involving multiple random variables in a higher-order-logic theorem prover, the linearity of expectation property has been formally verified in [19] as the following lemma.

Lemma 4 (Linearity of Expectation).

$$\vdash \forall\ \mathtt{L}.(\forall \mathtt{R}.(\mathtt{mem\ R\ L}) \Rightarrow ((\mathtt{R} \in \mathtt{indep}) \wedge (\mathtt{summable}(\lambda \mathtt{n}.\mathtt{n}\mathbb{P}\{\mathtt{s}|\mathtt{fst(R\ s)=n}\})))) \Rightarrow (\mathtt{expec\ (sum\_rv\_lst\ L)} = \sum_{\mathtt{n=0}}^{\mathtt{length\ L}} (\mathtt{expec\ (el\ (length\ L\ -\ (n+1))\ L)}))$$

The predicate `mem`, in the above assumption, is defined in the HOL *list* theory and it accepts an element and a list and returns `True` if the given element belongs to the given list. Thus, the assumption in the above theorem ensures that all random variables in the given list `L` preserve *strong function independence*, i.e., they $\in$ `indep`, and the infinite summations in their corresponding expectation definitions converge to a well-defined value (using the `summable` function explained in Table 1). The function `length`, defined in the HOL *list* theory, and used in the conclusion of the above theorem returns the length of its list argument and the function `el`, also defined in the *list* theory, accepts a positive integer number, say `n`, and a list and returns the `n`-th element of the given list. Whereas the HOL function `sum_rv_lst`, defined in [19] provides the summation of all random variables in the given list of random variables. Thus, the left-hand-side (LHS) of the conclusion of Lemma 4 represents the expectation of the summation of a list L of random variables. Whereas, the right-hand-side (RHS) of the conclusion of Lemma 4 represents the summation of expectations of all elements in the same list L.

Next, we illustrate the utilization of the above mentioned higher-order-logic foundations for the formalization of *indicator random variables*, which facilitates conducting the expected time complexity analysis of algorithms in the HOL theorem prover.

5 Formalization of the Indicator Random Variable

An indicator random variable is a special kind of random variable associated with the occurrence of an event. The indicator random variable I_A associated with an event A is usually defined as follows:

$$I_A = \begin{cases} 1, & \text{if the event } A \text{ occurs,} \\ 0, & \text{otherwise.} \end{cases} \qquad (2)$$

In other words, I_A maps all outcomes in the set A to 1 and all outcomes outside A to 0. Indicator random variables are the fundamental building blocks of many probability distributions. Moreover, they exhibit many useful characteristics and allow a convenient method for converting between probabilities and expectations[18]. Because of these features, they are quite frequently used in the probabilistic analysis of algorithms.

Based on the approach described in the previous section, the indicator random variable can be formalized in higher-order logic as the following function.

Definition 2 (Indicator Random Variable).

```
⊢ ∀ p.  ind_rv p = bind (ber_rv p)
   (λa.  unit(if a then 1 else 0)).
```

The above definition models an indicator random variable that is associated with an event with occurrence probability `p`. For this purpose, it utilizes the formal definition of the Bernoulli random variable (`ber_rv`), which is described in the previous section. The indicator random variable function `ind_rv` accepts the occurrence probability `p` as a real number and returns the corresponding indicator random variable value as a *positive integer*, which could either be a `1` or a `0`.

In order to ensure the correctness of the formal definition of the indicator random variable as well as to facilitate its utilization for the analysis of algorithms, we formally verify the following properties for it.

Theorem 1 (PMF for the Indicator R.V.).

```
⊢ ∀ p.  0 ⩽ p ∧ p ⩽ 1 ⇒
   ℙ {s | fst(ind_rv p s) = 1} = p.
```

Theorem 2 (Expectation for the Indicator R.V.).

```
⊢ ∀ p.  0 ⩽ p ∧ p ⩽ 1 ⇒
   expec(λs.  ind_rv p s) = p.
```

The formal proofs for the above properties are based on the PMF and expectation relations for the Bernoulli random variable, given in Lemmas 1 and 3, respectively, along with some basic arithmetic and set theoretic reasonings. It is important to note here that both of these theorems are verified under the assumption that `p` lies in the interval `[0, 1]`, which is the allowed range for a probability. According to Theorem 2, the expectation of an indicator random variable is equal to its occurrence probability. This property simplifies the expectation analysis significantly and thus is one of the main strengths of analyzing algorithms by modeling them in terms of indicator random variables.

Indicator random variables have been found to be quite useful for modeling algorithms in which we perform repeated trials as each one of such trials can be modeled as an indicator random variable. Thus in order to facilitate the higher-order-logic formalization and analysis of such algorithms, we now define a function that models a collection of indicator random variables as a list.

Definition 3 (List of Indicator R.Vs.).

```
⊢ (∀ ps.  ind_rv_list 0 ps = []) ∧
   ∀ n ps.  ind_rv_list(n + 1) ps
   = ind_rv_list n ps ++ [ind_rv (ps n)].
```

The HOL operator `++` in the above definition represents the list append operation. The recursive function `ind_rv_list` accepts a *positive* integer number `n` and a sequence of probabilities `ps` with data type (positive integer → real) and returns a list of `n` indicator random variables with respective probabilities from the sequence `ps`. Thus, if the function `ind_rv_list` is called with arguments $\mathtt{ps} = \langle \mathtt{p_0}, \mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_{n-1}} \rangle$ and `n` then it would return a list of indicator random variables $[\mathtt{I(p_0)}; \mathtt{I(p_1)}; \mathtt{I(p_2)}; \ldots; \mathtt{I(p_{n-1})}]$, where, `I(p)` represents an indicator random variable with success probability `p`. It is important to note that the usage of a sequence of probabilities in the above definition provides us with the flexibility to construct a list of indicator random variables with distinct probabilities.

Next, we formally verify the following very useful relationship regarding the expectation of a list of indicator random variables.

Theorem 3 (Expectation for the Indicator R.V. List).

$$\vdash \forall\ \texttt{n ps.}(\forall\ \texttt{i.}0 \leqslant (\texttt{ps i}) \wedge (\texttt{ps i}) \leqslant 1)$$
$$\Rightarrow \texttt{expec(sum\_rv\_lst(ind\_rv\_list n ps))}$$
$$= \texttt{sum}(0,\ \texttt{n})(\lambda\texttt{i. ps i}).$$

The assumption in the above theorem ensures that all real values in the probability sequence `ps` are bounded in the interval `[0, 1]` as this is the allowed range for a probability. According to Theorem 3, the expectation of the summation of all random variables in the list of random variables obtained by calling the function `ind_rv_list` with parameters n and *ps* is equal to the summation of all corresponding probabilities in the probability sequence *ps*. This result is quite important as it allows us to simplify a complex problem of evaluating the expectation of a sum of random variables to a simple summation of probability terms.

We proceed with the verification of Theorem 3 by rewriting its LHS using the linearity of expectation property, given in Lemma 4, as follows:

$$\sum_{\texttt{i}=0}^{\texttt{n}-1}\texttt{Ex}\ [\texttt{el}\ (\texttt{n}-(\texttt{i}+1))\ (\texttt{ind\_rv\_list n ps})]$$
$$=\sum_{\texttt{i}=0}^{\texttt{n}-1}(\texttt{ps i}). \tag{3}$$

The above mentioned substitution became possible because all random variables in the list of random variables generated by the function `ind_rv_list` satisfy the preconditions for Lemma 4, i.e., they preserve *strong function independence* because all of them are indicator random variables and thus access the infinite Boolean sequence using `bind` and `unit` operators only, as illustrated in Definition 3, and their corresponding expectations are well-defined as given in Theorem 2. Another simplification that has been made in the above substitution is the replacement of the term (`length (ind_rv_list n ps)`), which appears in the RHS of the linearity of expectation property, by the number n. The justification for this simplification was also formally verified in the HOL theorem prover.

The next step in proving Theorem 3 is to rewrite the LHS of (3) as follows:

$$\sum_{\texttt{i}=0}^{\texttt{n}-1}(\texttt{ps}(\texttt{n}-(\texttt{i}+1))) = \sum_{\texttt{i}=0}^{\texttt{n}-1}(\texttt{ps i}) \tag{4}$$

since the expectation of the i-th indicator random variable in the list (`ind_rv_list n ps`) can be proved to be equal to (`ps i`) using the result of Theorem 2 and some basic list properties. Finally, (4) can be verified based on arithmetic reasoning and the properties of summation, which also completes the HOL proof of Theorem 3.

Many computation algorithms can be simply described as a summation of indicator random variables for their expected time complexity analysis. Theorem 3 plays an important role in conducting their expected time complexity analysis in a theorem prover, as it allows us to transform the verification problem of an expectation relation to a verification involving a simple summation over real numbers. As an example, we present the analysis of the *hat-check* problem in the next section. Besides being useful for the analysis of this specific class of algorithms, which can be described as a simple summation of indicator random variables, the indicator random variable approach can also be utilized for the analysis of more complex algorithms. In order to illustrate the utilization and effectiveness of the proposed approach for other kinds of algorithms, we also present, in the next section, the analysis of the *birthday paradox* and the *hiring problem*, which we believe to be representative to many algorithms frequently used in computer science.

6 Applications

6.1 Hat-Check Problem

The hat-check problem is a classic combinatorial question: there is a dinner party where n gentlemen check their hats. The hat-check girl absentmindedly throws the claim checks away rather than putting them with the hats. When the gentlemen return for their hats, the hat-check girl returns them randomly. What is the number of gentleman who get their own hat back? The algorithm for this problem is a simple counting one and is given below.

Algorithm 1. Hat-Check Problem

Input: number of gentlemen in the party n

Output: number of gentlemen that were able to acquire their hats after the party x

```
$x \leftarrow 0$
for $i \leftarrow 1$ to $n$
    do if gentleman $i$ has his own hat
        then $x \leftarrow (x + 1)$
```

It samples all the n gentlemen in the party and counts the ones that were able to acquire their own hat. But the implementation of this algorithm for analysis purposes is not a very straightforward task because of the unpredictable nature of the input, i.e., the input could be any one of the 2^n possible combinations of n men with either their own or others hats. Thus, probabilistic techniques are applied. We assume that the hats are distributed uniformly among the men, i.e., the probability of any man in the party to get his own hat is the same $(1/n)$, and we find the expected number of people who acquire their own hats. By the definition of expectation, we have

$$Ex[X] = \sum_{k=0}^{\infty} kPr(X = k) \tag{5}$$

where Pr denotes the probability in the above equation. Again, evaluating the term $Pr(X = k)$, where X denotes the number of people who acquire their own hat, above is very cumbersome as this requires the probability of each permutation. The summation over this distribution would be even more complicated to solve.

The indicator random variable approach, described earlier, provides a very straightforward solution to the evaluation of the above expectation property. For each man i of the n men in the party, where $0 \leqslant i < n$, we define the indicator random variable X_i as follows:

$$X_i = \begin{cases} 1, & \text{if man } i \text{ acquires his own hat,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

The occurrence probability of the above indicator variable is $1/n$ because of the uniform distribution of hats. Now, the number of men that get their own hat is the sum of these indicator random variables

$$X = \sum_{i=0}^{n-1} X_i. \tag{7}$$

Based on the infrastructure presented in Section 5, the above equation can be formalized in higher-order logic as the following higher-order-logic function.

Definition 4 (Hat-Check Problem).

$$\vdash \forall\ \mathtt{n}.\ \ \mathtt{hchkp\ n} = \mathtt{sum\_rv\_lst(ind\_rv\_listn(}\lambda\mathtt{i.}\frac{\mathtt{1}}{\mathtt{n}}\mathtt{))}.$$

The function `hchkp` accepts a positive integer n, which represents the number of the men in the party that checked their hats and it returns the total number of men that were able to acquire their own hats. It utilizes the function `ind_rv_list` for this purpose, which models a list of indicator random variables and is given in Definition 3.

The next step after the formalization of the algorithm is to conduct its analysis in the theorem prover. For this purpose, we verify the following expectation property.

Theorem 4 (Hat-Check Expectation).

```
⊢ ∀ n.  0 < n ⇒
  (expec (hchkp n) = 1).
```

The assumption in the above theorem ensures that the number of men in the party are more than `0`. We proceed with the verification of the above theorem by utilizing the result of Theorem 3 to simplify it as follows:

$$\sum_{\mathtt{i}=0}^{\mathtt{n}-1} \left(\frac{\mathtt{1}}{\mathtt{n}}\right) = \mathtt{1}. \tag{8}$$

The above subgoal can now be discharged from the HOL goal stack using the basic properties of real summation along with some simple arithmetic reasoning. This also concludes the proof of Theorem 4.

According to Theorem 4, one man would be able to get his own hat back on average. The higher-order-logic formalization and analysis for the algorithm for the hat-check problem was very straightforward mainly because we were able to build upon the existing results like Definition 3 and Theorem 3 that were presented in the last section. Algorithms for many other commonly known problems, which can be expressed as a simple summation of indicator random variables like the Chinese appetizer problem[20], can also be analyzed in a similar way.

6.2 Birthday Paradox

The birthday paradox[18] or the birthday problem refers to the problem of determining the probability that in a randomly selected group of k people, two or more have the same birthday. Besides being an entertaining example, the birthday problem is one of the most famous problems in combinatorial probability and computer security applications.

The algorithm for the birthday paradox is given below. All we need to do is to pick each person from the group one by one and compare his or her birthday with all the persons in the group that have not been picked before, and keep track of the number of pairs with same birthdays. But as with all algorithm analysis problems, the input to this algorithm is unpredictable since we could be dealing with any group of people with birthdays distributed anywhere in the 365 days of the year. Hence, probabilistic techniques are used to model this random phenomenon and conduct the expected time complexity analysis of this problem.

Algorithm 2. Birthday Paradox

Input: number of persons in the group k

Output: number of pairs of persons with the same birthdays x

```
$x \leftarrow 0$
for  $i \leftarrow 1$ to $k$
    do for $j \leftarrow i + 1$ to $k$
        do if $j$ has the same birthday as $i$
            then $x \leftarrow (x + 1)$
```

As in most mathematical problems, we first have to make some simplifying assumptions, and then find the natural mathematical home for the problem. First of all, we ignore the issue of leap years and assume that all years have the same number of days, say n. Next, and most importantly, we assume that birthdays are more or less uniformly distributed across the n days of the year. Thus, the probability that a person's birthday falls on any particular day of the year is equal to $1/n$ and is also the same for any other day of the year. Lastly, we also assume that the birthdays of the k people in the group are distributed independently, i.e., the birthday of one person does not effect the birthday of any other person in the group in any way. All the above three assumptions are pretty reasonable based on our given conditions since the human birthdays are usually independent of one another and are uniformly distributed throughout the year[18].

Before we embark upon the higher-order-logic formalization of the algorithm for the birthday problem, we first formally analyze the probability for having a pair with matching birthdays, usually termed as the probability of success for the birthday paradox. This probability can be expressed formally as follows:

Definition 5 (Success Probability for the Birthday Paradox).

$$\begin{aligned}&\vdash \forall\ \mathtt{n}.\ \ \mathtt{bdayp\_suc\_prob\ n} = \\ &\quad \mathbb{P}\ \{\mathtt{s}\ \mid\ \mathtt{fst(unif\_rv\ n\ s)} = \\ &\quad \mathtt{fst(unif\_rv\ n\ (snd(unif\_rv\ n\ s)))}\}.\end{aligned}$$

According to the above definition, the probability of success for the birthday problem with n days a year is equal to the probability of the event when two independent Uniform(n) random variables generate the same values. The two Uniform(n) random variables in the above definition correspond to the birthdays of two persons in a group based on the above mentioned assumptions. The independence between the two Uniform(n) random variables is ensured because of the fact that the second uniform random variable on the RHS of the equality utilizes the remaining portion of the infinite Boolean sequence from the first Uniform(n) random variable that is on the LHS of the equality.

Next, we formally verify that the success probability for the birthday paradox is equal to $1/n$. Thus, the probability of having the same birthday for two persons in the group is the same as the probability that the birthday of one of them falls on a given day. The corresponding higher-order-logic theorem can be expressed as follows:

Theorem 5 (Success Probability for the Birthday Paradox).

$$\begin{aligned}&\vdash \forall\ \mathtt{n}.\ \ 0 < \mathtt{n} \Rightarrow \\ &\quad (\mathtt{bdayp\_suc\_probn} = \frac{1}{\mathtt{n}}).\end{aligned}$$

The assumption in the above theorem ensures that the value of n, i.e., the number of days in the year, is greater than 0 since without this assumption there is no point in analyzing the birthday problem. Also, it allows us to remove the division by 0 problem for the RHS term in the above theorem. We proceed with the verification by first rewriting with the definition of the function `bdayp_suc_prob` and simplifying the set that appears on the LHS as follows:

$$\begin{aligned}&\mathbb{P}(\bigcup_{\mathtt{i}<\mathtt{n}}(\{\mathtt{s}\mid\mathtt{fst\ (unif\_rv\ n\ s)} = \mathtt{i}\}\cap \\ &\quad\{\mathtt{s}\mid\mathtt{fst\ (unif\_rv\ n} \\ &\quad\mathtt{(snd\ (unif\_rv\ n\ s)))} = \mathtt{i}\})) = \frac{1}{\mathtt{n}}.\end{aligned}\tag{9}$$

Now, using the additive probability law $((A \cap B = \varnothing) \Rightarrow (\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)))$, the above subgoal can be further simplified as follows:

$$\begin{aligned}&\sum_{\mathtt{i}=0}^{\mathtt{n}-1}(\mathbb{P}(\{\mathtt{s}\mid\mathtt{fst\ (unif\_rv\ n\ s)} = \mathtt{i}\}\cap \\ &\quad\{\mathtt{s}\mid\mathtt{fst\ (unif\_rv\ n} \\ &\quad\mathtt{(snd\ (unif\_rv\ n\ s)))} = \mathtt{i}\})) = \frac{1}{\mathtt{n}}.\end{aligned}\tag{10}$$

The above subgoal can be further simplified using the independence property between the two uniform random variables, the product law of probability $(\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B))$ and the PMF of the uniform random variable, given in Lemma 2, as follows:

$$\sum_{\mathtt{i}=0}^{\mathtt{n}-1}\left(\frac{1}{\mathtt{n}}\right)\left(\frac{1}{\mathtt{n}}\right) = \frac{1}{\mathtt{n}}.\tag{11}$$

This subgoal can now be verified based on the properties of real summation, which also concludes the proof for Theorem 5.

The probability of success for the birthday paradox can now be utilized to formalize the algorithm for its probabilistic analysis using the proposed indicator random variable approach. For each pair (i, j) of the k people in the group, where $0 \leqslant i \leqslant j < k$, we define the indicator random variable X_{ij} as follows:

$$X_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ have the same birthday,} \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The occurrence probability of the above indicator variable is $1/n$ as has been already verified in Theorem 5. The algorithm for the birthday paradox is a simple counting algorithm that counts the number of pairs of individuals, present in the given group, having the same birthday. This algorithm can now be developed in terms of the indicator random variables as the one that counts the values of the indicator random variable for all possible pair combinations in the given group is as follows:

$$X = \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} X_{ij}. \tag{13}$$

The formalization of the above algorithm cannot be done using the summation of a list of indicator random variables as was the case in the hat-check problem. So, we formalize it, based on the infrastructure presented in Section 5, using the following two recursive functions.

Definition 6 (Birthday Paradox).

$$\begin{aligned}
&\vdash (\forall\ \texttt{n.}\ \ \texttt{bdayp\_helper 0 n = unit 0})\\
&\quad (\forall\ \texttt{k n.}\ \ \texttt{bdayp\_helper(k + 1) n =}\\
&\quad\ \texttt{bind(bdayp\_helper k n)}\\
&\quad (\lambda\texttt{a.}\ \ \texttt{bind(ind\_rv}\ (\tfrac{1}{\texttt{n}}))\\
&\quad\ (\lambda\texttt{b.}\ \ \texttt{unit(b + a))))}
\end{aligned}$$

$$\begin{aligned}
&\vdash (\forall\ \texttt{n.}\ \ \texttt{bdayp 0 n = unit 0})\\
&\quad (\forall\ \texttt{k n.}\ \ \texttt{bdayp(k + 1) n =}\\
&\quad\ \texttt{bind(bdayp k n)}\\
&\quad (\lambda\texttt{a.}\ \ \texttt{bind(bday\_helper k n)}\\
&\quad\ (\lambda\texttt{b.}\ \ \texttt{unit(b + a)))).}
\end{aligned}$$

The functions `bdayp_helper` and `bdayp` model the inner and outer summations of (13), respectively. Whereas the function `ind_rv`, defined in Definition 2, models the indicator random variable given in (12) with occurrence probability $1/n$. The function `bdayp` accepts two parameters k and n, which represent the population of the group and the number of days in a year, respectively, and it returns the total number of pairs of individuals having the same birthday in the given group.

Now, for the expected time complexity analysis of this algorithm, we verify the following expectation property.

Theorem 6 (Birthday Paradox Expectation).

$$\vdash \forall\ \texttt{k n.}\ \ 0 < \texttt{n} \wedge 2 \leqslant \texttt{k} \Rightarrow$$
$$(\texttt{expec(bdayp k n)} = \frac{\texttt{k(k} - 1)}{2\texttt{n}}).$$

The assumptions in the above theorem ensure that the number of days in a year are more than 0 and the population is at least 2 or more in order to have 1 pair at minimum. We proceed with the verification of the above theorem by performing induction on the variable k, which generates the following two subgoals.

$$\texttt{expec(bdayp 2 n)} = \frac{2}{2\texttt{n}} \tag{14}$$

$$\begin{aligned}
&2 \leqslant \texttt{k}\ \wedge (\texttt{expec(bdayp k n)} = \frac{\texttt{k(k} - 1)}{2\texttt{n}}) \Rightarrow\\
&\qquad \texttt{expec(bdayp(k} + 1)\ \texttt{n)} = \frac{(\texttt{k} + 1)\texttt{k}}{2\texttt{n}}.
\end{aligned} \tag{15}$$

The base case can be rewritten using the definition of the function `bdayp` as follows:

$$\texttt{expec}\Big(\lambda\texttt{s. ind\_rv}\Big(\frac{1}{\texttt{n}}\Big)\ \texttt{s}\Big) = \frac{2}{2\texttt{n}} \tag{16}$$

which can be simply verified using the expectation theorem for the indicator random variable as $1/n$ lies in the interval $(0, 1]$ when $0 < n$.

We proceed with the verification of the step case by first rewriting the `expec(bdayp(k+1) n)` part as follows:

$$\begin{aligned}
&2 \leqslant \texttt{k}\ \wedge (\texttt{expec(bdayp k n)} = \frac{\texttt{k(k} - 1)}{2\texttt{n}}) \Rightarrow\\
&\quad \texttt{expec(bdayp k n)} + \texttt{expec(bdayp\_helper k n)}\\
&\quad = \frac{(\texttt{k} + 1)\texttt{k}}{2\texttt{n}}.
\end{aligned} \tag{17}$$

This substitution is made based on the definition of the function `bdayp`, given in Definition 6, and the linearity of expectation property, given in Lemma 4. This allows us to utilize the second assumption in the subgoal as follows:

$$\begin{aligned}
&2 \leqslant \texttt{k}\ \Rightarrow\ \frac{\texttt{k(k} - 1)}{2\texttt{n}} + \texttt{expec(bdayp\_helper k n)}\\
&\quad = \frac{(\texttt{k} + 1)\texttt{k}}{2\texttt{n}}.
\end{aligned} \tag{18}$$

The above subgoal can now be further simplified by rearranging the terms along with some arithmetic reasoning as follows:

$$\texttt{expec(bdayp\_helper k n)}\ =\ \frac{\texttt{k}}{\texttt{n}}. \tag{19}$$

We verified the above mentioned subgoal, which represents the expectation of the function `bdayp_helper`, in a similar way as we handled the expectation property of the function `bdayp`, i.e., by using induction on variable k followed by using the linearity of expectation property and the expectation of the indicator random variable, given in Theorem 2, along with some arithmetic reasoning. The verification of the above subgoal also concludes the verification of Theorem 6.

Theorem 6 provides very useful insights into the birthday paradox. It can be clearly observed that the expected number of pairs of people with the same birthday would be at least 1 if $k(k-1) \geqslant 2n$. This means that if we have $\sqrt{2n}+1$ or more individuals in a room, then on average we can expect at least two people to have the same birthday. For $n = 365$, we need at least 28 people to have one pair of people to have the same birthday on average.

6.3 Hiring Problem

The hiring problem[18] is a combinatorial problem that captures a fundamental issue which arises in many applications where one must make decisions under uncertainty. In its most general form, the hiring problem concerns a company that wants to hire the best possible office assistant through an employment agency. Meanwhile, the company also needs an office assistant right away and it cannot wait for the best candidate to come along. So they decide to contact an employment agency, which has n candidates available for this job, and ask them to send a new candidate for interview every day. The company hires the first candidate to fill the vacant position for the office assistant but continues to interview new candidates. If a new applicant is found to be better qualified than the existing office assistant, he is hired and the existing office assistant is fired. The employment agency charges the company a small interviewing cost, say c_i, associated with each candidate interview and a comparatively large hiring cost, say c_h, associated with each hiring. The hiring problem is to find out the cost associated with this kind of a hiring strategy.

The cost algorithm for the hiring problem is given below. In the given hiring strategy, n candidates are always interviewed, irrespective of the number of people that are hired. Now, if we assume that a total number of m candidates get hired in the above strategy then the total cost associated with the algorithm would be $nc_i + mc_h$. In the worst case, each candidate that is interviewed is also hired and thus m becomes equal to n as well. This happens only if the candidates come in increasing order of quality. However, this does not always happen in practice and the candidates arrive in an unpredictable fashion. Therefore, probabilistic techniques are relied upon to evaluate the typical or average case cost for this algorithm.

Algorithm 3. Hiring Problem Cost

```
Input: number of available candidates $n$, hiring cost $c_h$,
       interviewing cost $c_i$
Output: hiring problem cost $c_{hp}$
$c_{hp} \leftarrow 0$
$best \leftarrow 0$
for  $i \leftarrow 1$ to $n$
    do interview candidate $i$
        $c_{hp} \leftarrow (c_{hp} + c_i)$
        if candidate $i$ is better than $best$
            then $best \leftarrow i$
                 hire candidate $i$
                 $c_{hp} \leftarrow (c_{hp} + c_h)$
```

For conducting the expected time complexity analysis of the hiring problem, we again utilize the proposed indicator random variable approach. For each candidate i, we define an indicator random variable as follows:

$$X_i = \begin{cases} 1, & \text{if candidate } i \text{ is hired,} \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

A candidate i is hired only if it is better than each of the already interviewed $i-1$ candidates. If we assume that the quality of candidate arrival is uniformly distributed then candidate i has a probability of $1/i$ of being hired or of being better than the already interviewed $i-1$ candidates. Thus, the occurrence probability of an indicator random variable corresponding to candidate i is equal to $1/i$ in the above mentioned indicator random variable.

Now, the hiring problem cost algorithm can be expressed as the following summation.

$$X = \sum_{i=0}^{n-1} c_i + c_h X_i. \tag{21}$$

The above equation does not represent a simple sum of indicator random variables and thus cannot be formalized using the function given in Definition 3. Thus, we formalized it with the following recursive function, based on the infrastructure presented in Section 5.

Definition 7 (Hiring Problem).

```
⊢ (∀ ch ci.  hirep 0 ch ci = unit 0)
   (∀ n ch ci.  hirep(n + 1) ch ci =
    bind(hirep k ch ci)
   (λa.  bind(ind_rv (1/n))
    (λb.  unit(if b = 1
           then (ch + ci + a)
           else (ci + a))))).
```

The function `hirep` accepts three parameters `n`, $\mathtt{c_h}$ and $\mathtt{c_i}$, which represent the number of available candidates, the cost of hiring and cost of interviewing a candidate, respectively. Whereas, it returns the total cost associated with the hiring strategy explained in this subsection. For this purpose, it utilizes the indicator random variable function `ind_rv`, defined in Definition 2, with occurrence probability $1/i$ for candidate number i.

The next step after the formalization of the algorithm is to conduct its analysis in the theorem prover. We verified the following expectation property in this regard.

Theorem 7 (Hiring Problem Expectation).

$$\vdash \forall\ \mathtt{n\ ch\ ci.}\ \ \mathtt{expec\ (hirep\ n\ ch\ ci)}$$
$$= \mathtt{n\ ci\ +\ ch} \sum_{\mathtt{i=0}}^{\mathtt{n-1}} \frac{1}{\mathtt{i+1}}.$$

We proceed with the verification of the above theorem by performing induction on the variable `n`, which generates the following two subgoals.

$$\mathtt{expec(hirep\ 0\ ch\ ci)} = \mathtt{ch}\sum_{i=0}^{0}\frac{1}{i+1} \tag{22}$$

$$\mathtt{expec(hirep\ n\ ch\ ci)} = \mathtt{nci} + \mathtt{ch}\sum_{i=0}^{n-1}\frac{1}{i+1} \Rightarrow$$
$$\mathtt{expec(hirep}\ (n+1)\ \mathtt{ch\ ci)}$$
$$= (n+1)\ \mathtt{ci}\ +\ \mathtt{ch}\sum_{i=0}^{n}\frac{1}{i+1}. \tag{23}$$

The base case can be simply verified based on the definitions of the expectation and the function `hirep`, given in Definitions 1 and 7. respectively, along with some arithmetic reasoning. For the verification of the step case, we first rewrite the `expec(hirep(n+1) ch ci)` part using the definition of the function `hirep`, given in Definition 7, and the linearity of expectation property, given in Lemma 4, as follows:

$$\mathtt{expec(hirep\ n\ ch\ ci)} = \mathtt{nci} + \mathtt{ch}\sum_{i=0}^{n-1}\frac{1}{i+1} \Rightarrow$$
$$\mathtt{ch\ expec}\Big(\mathtt{ind\_rv}\Big(\frac{1}{n+1}\Big)\Big) +$$
$$\mathtt{ci} + \mathtt{expec(hirep\ n\ ch\ ci)}$$
$$=(n+1)\ \mathtt{ci}\ +\ \mathtt{ch}\sum_{i=0}^{n}\frac{1}{i+1}. \tag{24}$$

Now the assumption, given in (24), can be used to obtain the following subgoal:

$$\mathtt{ch\ expec}\Big(\mathtt{ind\_rv}\Big(\frac{1}{n+1}\Big)\Big) +$$
$$\mathtt{ci} + \mathtt{nci} + \mathtt{ch}\sum_{i=0}^{n-1}\frac{1}{i+1}$$
$$= (n+1)\ \mathtt{ci}\ +\ \mathtt{ch}\sum_{i=0}^{n}\frac{1}{i+1}. \tag{25}$$

The above subgoal can now be verified by using the expectation property of the indicator random variable, given in Theorem 2, along with some arithmetic reasoning. This also concludes the proof for Theorem 7.

The term $\sum_{i=0}^{n-1}\frac{1}{i+1}$, which appears on the RHS of Theorem 7, is basically equal to `ln n`. Thus, according to Theorem 7, even though n people are interviewed only `ln n` of them are hired on average. This result means that the average or expected cost of the hiring problem is $O(c_h ln(n))$.

6.4 Discussion

The successful handling of the expected time complexity analysis of the hat-check problem, birthday paradox and hiring problem clearly demonstrates the effectiveness of the proposed indicator random variable based approach for formalizing probabilistic algorithms and conducting their analysis in a higher-order-logic theorem prover. It is worthwhile to mention here that the algorithm analysis results presented in this subsection are not something that is new and they have been known for quite some time now. The real contribution of the paper lies in demonstrating the ability to conduct the analysis of these algorithms precisely using a computer based tool. Due to the formal nature of the algorithm implementations and inherent soundness of theorem proving, we have been able to verify the expectation properties of interest regarding the given algorithms with 100% precision; a novelty which is not available in simulation. Similarly due to the high expressibility of higher-order logic, we have been able to verify generic properties that are valid for all values of algorithm inputs. The proposed approach is also superior than the paper-and-pencil analysis methods in a way as the chances of making human errors, missing critical assumptions and proving wrongful statements are almost nil since all proof steps are applied within the sound core of the HOL theorem prover. These additional benefits come at the cost of the time and effort spent, while constructing the formal model of the algorithm and formally reasoning about its properties, by the user. But, the analysis infrastructure, presented and developed in Sections 4 and 5 of this paper, led to a significant reduction in the interactive verification effort. The analysis presented here for the three algorithms consumed around 1500 lines of HOL code and approximately 100 man hours.

7 Conclusions

In this paper, we utilized the mathematical probability theory formalized in a higher-order-logic theorem prover to develop a formal expected time complexity analysis approach for algorithms. The main idea behind this approach is to construct a higher-order-logic model of the algorithm along with its random components and to verify the corresponding performance characteristics and computation complexity relations in a theorem prover. We specifically targeted algorithms that can be modeled using indicator random variables and thus also presented a higher-order-logic definition of the indicator random variable as well as the formal verification of some of its key properties. Because of the formal nature of the models in the proposed approach, the probabilistic analysis is free of approximation and precision errors, and due to the high expressive nature of higher-order logic a wider range of algorithms can be analyzed. Thus, the theorem proving based expected time complexity analysis approach can prove to be very useful for algorithms used in safety critical and highly sensitive engineering and scientific applications.

The proposed approach was used to conduct the analysis of algorithms for three well-known problems, i.e., the hat-check problem, the birthday paradox and the hiring problem. We developed higher-order-logic based formal models for these algorithms, based on which we formally verified the expectation relations of some of their key characteristics. The formal definition of the indicator random variable and its formally verified properties greatly helped us to speed up the analysis process. The results obtained are 100% precise and confirmed the results of paper-and-pencil based analysis approaches. The successful handling of these real-world algorithm analysis problems by the proposed approach clearly demonstrates its feasibility for other algorithm analysis problems. To the best of our knowledge, this is the first study on using higher-order-logic theorem proving for the expected time complexity analysis of such algorithms.

The proposed probabilistic approach can be readily applied for the analysis of many other algorithms, such as, the *balls and bins* problem[18], the *longest streak of heads* problem[18], the *on-line hiring* problem[18] the *Chinese appetizer* problem[20] and the *Quicksort* algorithm[5]. Similarly, besides the expectation properties, we can also verify other statistical properties like variance and tail distribution bounds regarding the algorithm characteristics using the formalizations presented in [19].

The time complexity of an algorithm is basically the time that it takes to run in terms of its inputs. A commonly used metric for calculating time complexities of algorithms is the *big O notation*, where the main idea is to remove all multiplicative constant factors and lower order terms from the time complexity relations. The *big O notation* method is quite useful in computing the time complexities of algorithms as their input size becomes very very large. In this paper, we presented an approach to formally estimate the average time complexities of algorithms. Based on these foundations, we can also formally analyze the time complexities of algorithms as their inputs become very large or possibly infinite in a higher-order-logic theorem prover. We are working towards this goal by building upon the higher-order-logic formalization of limit of a real sequence[58].

References

[1] Knuth D E. The Art of Computer Programming. Addison-Wesley Professional, 1997.

[2] Whittle P. Probability via Expectation. Springer, 2000.

[3] Kozen D. A probabilistic PDL. *Journal of Computer and System Sciences*, 1985, 30(2): 162-178.

[4] Jones C. Probabilistic Non-Determinism [Ph.D. Dissertation]. University of Edinburgh, Edinburgh, UK, 1990.

[5] Mitzenmacher M, Upfal E. Probability and Computing. Cambridge University Press, 2005.

[6] Whitney M . Exploring the birthday paradox using a Monte Carlo simulation and graphing calculators. *Mathematics Teacher*, 2001, 94(4): 258-262.

[7] Hastingsr K . Introduction to Probability with Mathematica. Chapman and Hall/CRC, 2000.

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

[9] MacKay D J C. Introduction to Monte Carlo methods. In *NATO Advanced Study Institute on Learning in Graphical Models*, Erice, Italy, 1998, pp.175-204.

[10] Flajolet P, Salvy B, Zimmermann P. Automatic average-case analysis of algorithms. *Theoretical Computer Science*, 1991, 79(1): 37-109.

[11] Adams A, Gottliebsen H, Linton S A, Martin U. Automated theorem proving in support of computer algebra: Symbolic definite integration as a case study. In *Proc. Symbolic and Algebraic Computation*, Vancouver, Canada, July 28-31, 1999, pp.253-260.

[12] Hall A. Realising the benefits of formal methods. *J. Universal Computer Science*, 2007, 13(5): 669-678.

[13] Clarke E, Grumberg O, Long D. Verification tools for finite state concurrent systems. In *Proc.REX School/Symp. A Decade of Concurrency — Reflections and Perspectives*, Noordwijkerhout, The Neitherlands, Jun. 1-4, 1993, pp.124-175.

[14] Harrison J. Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, 2009.

[15] Hurd J. Formal verification of probabilistic algorithms [Ph.D. Dissertation]. University of Cambridge, Cambridge, UK, 2002.

[16] McIver A K, Morgan C C. Abstraction, Refinement and Proof for Probabilistic Systems. Spriger, 2005.

[17] Hurd J, McIver A, Morgan C. Probabilistic guarded commands mechanized in HOL. *Theoretical Computer Science*, 2005, 346(1): 96-112.

[18] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. The MIT Press, 2001.

[19] Hasan O, Tahar S. Using theorem proving to verify expectation and variance for discrete random variables. *Journal of Automated Reasoning*, 2008, 41(3/4): 295-323.

[20] Grinstead C M, Snell J L. Introduction to Probability. American Mathematical Society, 1997.

[21] Mckinney E H. Generalized birthday problem. *The American Mathematical Monthly*, 1966, 73(4): 385-387.

[22] de Montmort P R. Essay d'Analyse sur les Jeux de Hazard. Published anonymously, 1708.

[23] Euler L. Calcul de la probabilite dans le jeu de rencontre. *Memoires de lAcademie des Sciences de Berlin*, 1753, (7): 255-270.

[24] P S de Laplace. Theorie Analytique des Probabilites. Published anonymously, 1812.

[25] Takacs L. The problem of coincidences. *Archive for History of Exact Sciences*, 1980, 3(21): 229-244.

[26] Akutsu T. On determining the congruity of point sets in higher dimensions. In *Proc. International Symposium on Algorithms and Computation*, Beijing, China, Aug. 25-27, 1994, pp.38-46.

[27] Flajolet P, Gardy D, Thimonier L. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 1992, 39(3): 207-229.

[28] Gazit H, Reif J H. A randomized parallel algorithm for planar graph isomorphism. *Journal of Algorithms*, 1998, 28(2): 290-314.

[29] Stinson D R. Cryptography, Theory and Practice. CRC Press, 2006.

[30] Gardner M. Mathematical games. *Scientific American*, 1960, 202: 150-153.

[31] Freeman P R. The secretary problem and its extensions: A review. *International Statistical Review*, 1983, 51(2): 189-206.

[32] Kleinberg R. A multiple-choice secretary algorithm with applications to online auctions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, Vancouver, Canada, Jan. 23-25, 2005, pp.630-631.

[33] Babaioff M, Immorlica N, Kleinberg R. Matroids, secretary problems, and online mechanisms. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, USA, Jan. 7-9, 2007, pp.434-443.

[34] Broder A Z, Kirsch A, Kumar R, Mitzenmacher M, Upfal E, Vassilvitskii S. The hiring problem and lake wobegon strategies. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, USA, Jan. 20-22, 2008, pp.1184-1193.

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

[36] Paulson L C. Isabelle: A Generic Theorem Prover. Springer, 1994.

[37] CoQ. http://pauillac.inria.fr/coq/, 2009.

[38] PVS. http://pvs.csl.sri.com, 2009.

[39] Nedzusiak A. σ-fields and probability. *Journal of Formalized Mathematics*, 1989, 1.

[40] Bialas J. The σ-additive measure theory. *Journal of formalized Mathematics*, 1990, 2.

[41] Hasan O, Tahar S. Formalization of the continuous probability distributions. In *Proc. Int. Conf. Automated Deduction*, Bremen, Germany, Jul. 17-20, 2007, pp.3-18.

[42] Hasan O, Tahar S. Performance analysis and functional verification of the stop-and-wait protocol in HOL. *Journal of Automated Reasoning*, 2009, 42(1): 1-33.

[43] Hasan O, Tahar S. Performance analysis of ARQ protocols using a theorem prover. In *Proc. International Symposium on Performance Analysis of Systems and Software*, Austin, USA, April 20-22, 2008, pp.85-94.

[44] Hasan O, Tahar S. Performance analysis of wireless systems using theorem proving. In *Proc. the First International Workshop on Formal Methods for Wireless Systems*, Toronto, Canada, Aug. 19-22, 2008, pp.3-18.

[45] Hasan O, Abbasi N, Tahar S. Formal probabilistic analysis of stuck-at faults in reconfigurable memory arrays. In *Proc. Int. Conf. Integrated Formal Methods*, Düsseldorf, Germany, Feb. 16-19, 2009, pp.277-291.

[46] Baier C, Haverkort B, Hermanns H, Katoen J P. Model checking algorithms for continuous time Markov chains. *IEEE Transactions on Software Engineering*, 2003, 29(4): 524-541.

[47] Rutten J, Kwaiatkowska M, Normal G, Parker D. Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. Vol.23 of CRM Monograph Series, American Mathematical Society, 2004.

[48] Baier C, Katoen J P. Principles of Model Checking. MIT Press, 2008.

[49] L de Alfaro. Formal verification of probabilistic systems [Ph.D. Dissertation]. Stanford University, Stanford, USA, 1997.

[50] Parker D. Implementation of symbolic model checking for probabilistic system [Ph.D. Dissertation]. University of Birmingham, Birmingham, UK, 2001.

[51] Kwiatkowska M, Norman G, Parker D. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 2005, 153(2): pp.5-31.

[52] Sen K, Viswanathan M, Agha G. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Proc. IEEE International Conference on the Quantitative Evaluation of Systems*, Torino, Italy, Sept. 19-22, 2005, pp.251-252.

[53] Norman G. Validation of Stochastic Systems: A Guide to Current Research, vol. 2925 of LNCS (Tutorial Volume), Chapter Analyzing Randomized Distributed Algorithms, Springer, 2004, pp.384-418.

[54] Church A. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940, 5: 56-68.

[55] Milner R. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 1977, 17: 348-375.

[56] Paulson L C. ML for the Working Programmer. Cambridge University Press, 1996.

[57] Harrison J. Formalized mathematics. Technical Report 36, Turku Centre for Computer Science, Finland, 1996.

[58] Harrison J. Theorem Proving with the Real Numbers. Springer, 1998.

Osman Hasan received the B.Eng. (Hons.) degree from the N-W.F.P University of Engineering and Technology, Pakistan, in 1997, and the M.Eng. and Ph.D. degrees from Concordia University, Montreal, QC, Canada, in 2001 and 2008, respectively. He served as an ASIC design Engineer from 2001 to 2003 in the industry prior to joining Concordia University in 2004 for his Ph.D. Currently, he is a research associate at the Hardware Verification Group, Concordia University. His current research interests include formal methods, higher-order-logic theorem proving and probabilistic analysis.

Sofiène Tahar received the Diploma degree in computer engineering from the University of Darmstadt, Germany, in 1990, and the Ph.D. degree with distinction in computer science from the University of Karlsruhe, Germany, in 1994. Currently, he is a professor and research chair in formal verification of system-on-chip at the Department of Electrical and Computer Engineering, Concordia University. His research interests are in the areas of formal hardware verification, system-on-chip verification, analog and mixed signal circuits verification, and probabilistic, statistical and reliability analysis of systems. Dr. Tahar, a professional engineer in the Province of Quebec, is the founder and director of the Hardware Verification Group at Concordia University. In 2007, he was named University Research Fellow upon receiving Concordia University' s Senior Research Award.