Formalization of Asymptotic Notations in HOL4

Nadeem Iqbal*, Osman Hasan*, Umair Siddique* and Falah Awwad**
*School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST), Islamabad, Pakistan
**College of Engineering, UAE University, Al Ain, UAE

Abstract—**Asymptotic notations characterize the limiting behavior of a function. They are extensively used in many branches of mathematics and computer science particularly in analytical number theory, combinatorics and computational complexity while analyzing algorithms. Traditionally, the mathematical analysis involving these notations has been done by paper-and-pencil proof methods or simulation. In order to introduce formal verification in this domain, this paper provides the higher-order-logic formalizations of O, Θ, Ω, o and ω notations and the formal verification of most of their classical properties of interest. The formalization is based on the theory of sets, real and natural numbers and has been done using the HOL4 theorem prover.**

I. Introduction

Traditionally, the computational time assessment of algorithms is done using benchmarking and asymptotic notations. In benchmarking, the main idea is to run the algorithms on a computer and then measure their speed in some time units. The benchmarking based assessment cannot be trusted completely because it measures the efficiency of a particular program which has been written in a specific language, running on a particular platform, with a particular compiler and a particular input data [1]. From this single benchmark, it is difficult to predict that how much time the algorithm will take if it is deployed on a system with a different set of specifications. This limitation can be overcome by using an analytical approach based on asymptotic notations [2]. The first asymptotic notation, i.e., the Big-O notation or simply O-notation was introduced by a number theorist Bachmann in 1894 [3]. In the following years, its properties and physical interpretation became more common, and it has been frequently used in algorithms analysis. Little-o notation was introduced by Landau in 1909 [4], and Big-Ω, Big-Θ, and Little-ω notations were presented by Knuth in 1976 [5].

The running time complexity analysis of algorithms has been traditionally done using paper-and-pencil proof methods or computer simulations (testing). Both of these techniques do not guarantee the accuracy of the analysis results. The paper-and-pencil proof methods are prone to human error and there is always a risk of forgetting to pen down a critical assumption besides the proof. For example, in 1973, Hopcroft and Tarjan [6] proposed a *linear time algorithm* to decompose a graph into its tri-connected components, which is a very crucial algorithmic step for graph-theoretic problems. However, their algorithmic description contains different flaws, which were discovered by Gutwenger and Mutzel [7] in 2001, when the highly complex algorithm was first implemented. Simulation, on the other hand, allows us to check that a property is true or not by analyzing its behavior under a subset of all possible inputs.

Given the extensive usage of asymptotic analysis of algorithms in safety-critical systems, there is a dire need of using formal methods support in this domain. We believe that higher-order-logic theorem proving [8] offers a promising solution for conducting formal asymptotic analysis of algorithms. The main reason being the highly expressive nature of higher-order logic, which can be leveraged upon to essentially model any system that can be expressed in a closed mathematical form. In fact, the asymptotic notations have been formalized in higher-order logic. The O, Ω and Θ notations have been formalized in Mizar based on non-negative sequences of real numbers [9] and asymptotic notions have been formalized in Isabelle/HOL [10], [11] using ring theory. In this paper, we mainly build a library of formalized asymptotic notations, i.e., O, Θ, Ω, o and ω, in the higher-order-logic theorem prover HOL4 [12] using the real number theory. The main motivation of this choice is to be able to utilize our formalized theory of asymptotic notations with the existing formalization of probability and information theories [13] in HOL4 to formally reason about probabilistic and information theoretic aspects of algorithm complexities. This combination would allow us to target the formal analysis of many safety-critical applications in the areas of information security, communication engineering and graph theoretic frameworks.

The rest of the paper is organized as follows: Section II presents a brief introduction to the HOL4 theorem prover. Then Section III provides our higher-order-logic definitions of the five asymptotic notations, i.e., O, Θ, Ω, o and ω. Next section (Section IV) provides the formal verification details about the properties of these notations using the HOL4 theorem prover. Finally, Section V concludes the paper.

II. HOL4 Theorem Prover

HOL4 is an interactive theorem prover developed by Mike Gordon at the University of Cambridge for conducting proofs in higher-order logic. It utilizes the simple type theory of Church [14] along with Hindley-Milner polymorphism [15] to implement higher-order logic.

A HOL4 theory is a collection of valid HOL4 types, constants, axioms and theorems, and is usually stored as a file in computers. Users can reload a theory in the HOL4 system and utilize the corresponding definitions and theorems right away. The concept of HOL4 theory allows 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.

HOL4 theories are organized in a hierarchical fashion. Any theory may inherit types, definitions and theorems from other available HOL4 theories. The HOL4 system prevents loops in this hierarchy and no theory is allowed to be an ancestor and descendant of a same theory. Various mathematical concepts have been formalized and saved as HOL4 theories by the HOL4 users. These theories are available to a user when he first starts a HOL4 session. We utilized the HOL4 theories of sets, positive integers and *real* analysis in our work. In fact, one of the primary motivations of selecting the HOL4 theorem prover for our work was to benefit from these built-in mathematical theories.

A. *Writing Proofs*

HOL4 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 sub-goals. 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 sub-goals. Some of these intermediate sub-goals 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 sub-goals and this concludes the proof for the desired theorem.

The HOL4 theorem prover includes many proof assistants and automatic proof procedures 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.

III. Formal Definitions of Asymptotic Notations

This section describes our formal definitions of asymptotic notations using higher-order logic. Some examples are also provided to facilitate understanding.

A. *The* O *Notation*

The *O*-notation provides an asymptotic upper bound for algorithms or functions. Its frequency of usage outnumbers the other notations because it gives us an upper bound on the complexity, i.e., it gives us a guarantee that at maximum a particular algorithm will take such a time for its execution. It can be defined for a given function g as follows:

Definition 1: *BigO Notation*
```
⊢ ∀ g. BigO (g:num → real) =
{(f:num → real)| (∃ c n_0.(∀ n. n_0 ≤ n
⟹ 0 < c ∧ 0 ≤ f(n) ≤ c * g(n)))}
```

Here `f` and `g` are functions which take a natural number `num` and return a real number `real`. The constants `c` and `n_0` are of type `real` and `num`, respectively. The `BigO` takes a function g as an input and returns the set of all functions f which qualify the condition $0 \leq \mathtt{f(n)} \leq \mathtt{c} * \mathtt{g(n)}$. For example,

$$n^2 + 20n + 100 = O(n^2)$$

means that there exist some positive constants c and $n0$ such that $n^2 + 20n + 100 \leq cn^2$. It is to be noted that "=" is not used to express "is equal to" but it refers to set membership in the context of asymptotic notations.

B. *The* Θ *Notation*

The Θ-notation is used when we need a more strict asymptotic bound than the one provided by *O*. It is defined as follows:

Definition 2: *BigTheta Notation*
```
⊢ ∀ g. BigTheta (g:num → real) =
{(f:num → real)|(∃ c1 c2 n_0.
(∀ n.n_0 ≤ n ⟹ 0 < c1 ∧ 0 < c2 ∧
0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n)))}
```

Here, `BigTheta` takes as an input a function g and returns the set of all functions f that satisfy a more strict condition, i.e.,$0 \leq \mathtt{c1} * \mathtt{g(n)} \leq \mathtt{f(n)} \leq \mathtt{c2} * \mathtt{g(n)}$ for all n greater than some $n0$ and for some real constants $c1$ and $c2$.

C. *The* Ω *Notation*

The Ω-notation is used when a lower asymptotic bound is required. It is defined for a function g as follows:

Definition 3: *BigOmega Notation*
```
⊢ ∀ g. BigOmega (g:num → real) =
{(f:num→real)| (∃ c n_0. (∀ n. n_0 ≤ n
⟹ 0 < c ∧ 0 ≤ c * g(n) ≤ f(n)))}
```

`BigOmega` also takes as an input function g and returns the set of all functions f which satisfy the condition $0 \leq \mathtt{c} * \mathtt{g(n)} \leq \mathtt{f(n)}$.

D. *The* o *Notation*

The *o*-notation is used to denote an upper bound that is not asymptotically tight. It is defined for a function g as follows:

Definition 4: *LittleO Notation*
```
⊢ ∀ g. LittleO (g:num → real) =
{(f:num → real)| (∃ c n_0.(∀ n. n_0 ≤ n
```

$\Longrightarrow$ 0 < c $\wedge$ 0 $\leq$ f(n) < c * g(n)))}

The *o*-notation is a variant of BigO. It takes as an input a function and returns the set of all functions which qualify the condition $0 \leq f(n) < c * g(n)$ for all values of n greater than n_0 and for a given real constant c.

E. The ω Notation

The ω-notation is used to denote a lower bound that is not asymptotically tight. It is defined for a function g as follows:

Definition 5: *LittleOmega Notation*
$\vdash$ $\forall$ g. LittleOmega (g:num $\rightarrow$ real) =
{(f:num$\rightarrow$real) | ($\exists$ c n_0.($\forall$ n. n_0 $\leq$ n
$\Longrightarrow$ 0 < c $\wedge$ 0 $\leq$ c * g(n) < f(n)))}

LittleOmega is analogous to BigOmega. It takes as an input a function g and returns the set of all functions f which satisfy the condition $0 \leq c * g(n) < f(n)$ for all values of n greater than n_0 and for a given real constant c.

IV. Formal Verification of Asymptotic Notations

In this section, we define and prove some of the key properties of asymptotic notations (O, Θ, Ω, o and ω), such as transitivity, product and transpose symmetry. Due to the fact that these properties have been formally verified ensures the correctness of our formal definitions.

A. The O Notation

We will discuss at length the formal proof details of a couple of key theorems of *O*-notation here:

Theorem 1: *Transitivity of O-Notation*
$\vdash$ $\forall$ f g h. f $\in$ (BigO g) $\wedge$ g $\in$ (BigO h)
$\Longrightarrow$ f $\in$ (BigO h)

Proof Sketch: We start the proof process of the above-mentioned transitivity property of the O-Notation by rewriting the above goal with the definition of BigO notation (Definition 1) along with some set-theoretic simplifications and we reach the following subgoal:

($\forall$ n. n_0 $\leq$ n $\Longrightarrow$ 0 < c $\wedge$ 0 $\leq$ f(n) $\wedge$
f(n) $\leq$ c * g(n)) $\wedge$
($\forall$ n. n_0′ $\leq$ n $\Longrightarrow$ 0 < c′ $\wedge$ 0 $\leq$ g(n)
$\wedge$ g(n) $\leq$ c′ * h(n)) $\Longrightarrow$
($\exists$ c n_0.$\forall$ n. n_0 $\leq$ n $\Longrightarrow$ 0 < c $\wedge$
0 $\leq$ f(n) $\wedge$ f(n) $\leq$ c * h(n))

At this stage, we have to provide specific values for the variables c and n_0. We specialized c and n_0 with $c * c'$ and MAX(n_0 n_0′), respectively. From the assumptions $0 < c$ and $0 < c'$, we can readily deduce that $0 < c * c'$. Furthermore, the function MAX returns maximum of the given pair of natural numbers. To fulfill the requirements in both the assumptions, we require such a value of n_0 that would fulfil both these conditions and that value is MAX(n_0 n_0′). Obviously, when the maximum of these two numbers will be less n, the other number will be automatically less than n too. By some straightforward arithmetic reasoning, we formally verified the given subgoal.

Theorem 2: *Sum of O-Notation*
$\vdash$ $\forall$ t1 t2 g1 g2.
t1 $\in$ (BigO g1) $\wedge$ t2 $\in$ (BigO g2) $\Longrightarrow$
(λn. t1 n + t2 n) $\in$ (BigO (max(g1, g2)))

Proof Sketch: We start the proof process of the above-mentioned sum of O-Notation property by rewriting the above goal with the definition of BigO notation (Definition 1) along with some set-theoretic simplifications and we reach the following subgoal:

($\forall$ n. n_0 $\leq$ n $\Longrightarrow$ 0 < c $\wedge$
0 $\leq$ t1(n) $\wedge$ t1(n) $\leq$ c * g1(n))$\wedge$
($\forall$ n. n_0′ $\leq$ n $\Longrightarrow$ 0 < c′ $\wedge$
0 $\leq$ t2(n) $\wedge$ t2(n) $\leq$ c′ * g2(n))
$\Longrightarrow$($\exists$ c n_0.$\forall$ n. n_0 $\leq$ n
$\Longrightarrow$ 0 < c $\wedge$ 0 $\leq$ (t1(n) + t2(n))$\wedge$
(t1(n) + t2(n)) $\leq$ c * max(g1(n), g2(n)))

Here for fulfilling the conditions of the assumptions $n\_0 \leq n$ and $n\_0' \leq n$, we again specialized n_0 by MAX(n_0, n_0′). Furthermore, we specialized c with $2 * \max(c, c')$. With some straightforward arithmetic reasoning, we proved our goal.

Moreover, we formally verified the following theorems of *O*-notation in HOL4 and the formal reasoning details can be found in our proof script [16].

1) $\vdash$ $\forall$ t1 t2 g1 g2.
t1 $\in$ (BigO g1) $\wedge$ t2 $\in$ (BigO g2) $\Longrightarrow$
(λn. t1 n * t2 n) $\in$ (BigO (g1 * g2))
2) $\vdash$ $\forall$ f g. f $\in$ (BigO g) $\Longrightarrow$
$\forall$ k. (λn. k * f n) $\in$ (BigO g)
3) $\vdash$ $\forall$ f. ($\exists$ n_0. ($\forall$m. n_0 $\leq$ m $\Longrightarrow$
0 $\leq$ f m) $\Longrightarrow$ f $\in$ (BigO f))

The first property in the above list is called the product property. If an algorithm having complexity t1 is run t2 times or vice versa, then the product of their complexities will still lie in the BigO of their product of orders of growth. The second property refers to a situation in which if an algorithm is run for k times, its overall complexity in this case will still lie in the same order of growth. The third property is a very interesting property of *O*-notation and is called reflexivity. It states that the complexity of an algorithm is always its own order of growth.

B. The Θ Notation

Theorem 3:
$\vdash$ $\forall$ f g.
f $\in$ (BigTheta g) $\Longleftrightarrow$ g $\in$ (BigTheta f)

Theorem 3 is the symmetry property of Θ-notation. We proceed with its verification by splitting the main goal into the following two subgoals:

```
⊢ ∀ f g.
 f ∈ (BigTheta g) ⟹ g ∈ (BigTheta f)
```

```
⊢ ∀ f g.
 g ∈ (BigTheta f) ⟹ f ∈ (BigTheta g)
```

The proof sketch for the first subgoal is given below and the other one was handled in a very similar way.

Proof Sketch: We start the proof process by rewriting the goal with the definition of BigTheta notation (Definition 2) along with some set-theoretic simplifications and we reach the following subgoal:

```
(∀ n. n_0 ≤ n ⇒ 0 < c1 ∧
  0 < c2 ∧ 0 ≤ c1 * g(n) ∧
  c1 * g(n) ≤ f(n) ∧ f(n) ≤ c2 * g(n))
  ⟹(∃ c1 c2 n_0.(∀ n. n_0 ≤ n) ⇒
  0 < c1 ∧ 0 < c2 ∧ 0 ≤ c1 * f(n)
 ∧ c1 * f(n) ≤ g(n) ∧ g(n) ≤ c2 * f(n))
```

Here again, the formal reasoning process relies on choosing the right set of values of variables `n_0`, `c1` and `c2`. We chose them to be `n_0`, `1/c2` and `1/c1`, respectively, and verified the subgoal based on arithmetic reasoning.

Moreover, we proved the transitivity, summation and reflexivity of BigTheta as the following theorems [16]:

```
1) ⊢ ∀ f g h.
     f ∈ (BigTheta g) ∧ g ∈ (BigTheta h)
     ⟹ f ∈ (BigTheta h)
2) ⊢ ∀ t1 t2 g1 g2.
      t1 ∈ (BigTheta g1) ∧ t2 ∈
    (BigTheta g2)
      ⟹ (λn. t1 n + t2 n) ∈ (BigTheta
    (g1 + g2))
3) ⊢ ∀f. (∃n_0. (∀m. n_0 ≤ m ⟹
    0 ≤ f m) ⟹ f ∈ (BigTheta f))
```

According to the second property, i.e., the sum property of BigTheta, if complexities of two algorithms lie in BigTheta, then the sum of their complexities will also lie in the corresponding sum of the orders of growth in the BigTheta.

C. The Ω Notation

Theorem 4:

```
⊢ ∀ t1 t2 g1 g2.
 t1 ∈ (BigOmega g1) ∧ t2 ∈ (BigOmega g2)
 ⟹ (λn. t1 n + t2 n) ∈
      BigOmega (min (g1, g2))
```

Theorem 4 implies that if an algorithm consists of two parallel components then the algorithm's overall efficiency will be determined by the part with a lower order of growth.

Proof Sketch: We start the proof process by rewriting the goal with the definition of BigOmega notation (Definition 3) along with some set-theoretic simplifications and we reached to the following goal:

```
(∀ n. n_0 ≤ n ⇒ 0 < c ∧
0 ≤ c * g1(n) ∧ c * g1(n) ≤ t1(n))∧
(∀ n. n_0′ ≤ n ⇒ 0 < c′
∧ 0 ≤ c′ * g2(n) ∧ c′ * g2(n) ≤ t2(n))
⟹( ∃ c n_0.∀ n. n_0 ≤ n ⇒
0 < c ∧ 0 ≤ c * min(g1(n), g2(n))
∧ c * min(g1(n), g2(n)) ≤ (t1(n) + t2(n))
```

Then we specialized the variables `c` and `n_0` with $2*\mathtt{min}(\mathtt{c},\mathtt{c}')$ and $\mathtt{MAX}(\mathtt{n\_0},\ \mathtt{n\_0}')$, respectively. This specialization allowed us to discharge all the conditions of the two assumptions and thus in turn prove the above subgoal using some arithmetic reasoning. Moreover, we proved the following theorems of Ω-notation [16]:

```
1) ⊢ ∀ f g h.
    f ∈ (BigOmega g) ∧ g ∈ (BigOmega h)
    ⟹ f ∈ (BigOmega h)
2) ⊢ ∀f. (∃n_0.
    (∀m. n_0 ≤ m ⟹ 0 ≤ f m)
    ⟹ f ∈ (BigOmega f))
3) ⊢ ∀ f g.
    f ∈ (BigO g) ⟹ g ∈ (BigOmega f)
```

D. The LittleO Notation

We verified the transitivity and transpose symmetry properties of the LittleO notation as the following higher-order-logic theorems [16].

```
1) ⊢ ∀ f g h.
    f ∈ (LittleO g) ∧ g ∈ (LittleO h)
    ⟹ f ∈ (LittleO h)
2) ⊢ ∀ f g.
    f ∈ (LittleO g) ⟺ g ∈
   (LittleOmega f)
```

E. The LittleOmega Notation

We verified the following properties of the LittleOmega notation as the following higher-order-logic theorems [16].

```
1) ⊢ ∀ f g.
    f ∈ (LittleOmega g) ∧ g ∈
   (LittleOmega h)
    ⟹ f ∈ (LittleOmega h)
2) ⊢ ∀ f g.
    f ∈ (LittleOmega g) ⟺ g ∈
   (LittleO f)
```

F. Asymptotic Complexity in Terms of Limits

In some situations, it is convenient to prove the complexity using the concept of limits. For example for a function $f(n) = n^2 + 20n + 100$, we can say that $f(n) \in O(n^2)$ by proving the following limit:$\lim_{n\to\infty} \frac{n^2+20n+100}{n^2} = 1$.

Indeed the above limit is the sufficient condition to prove that $f(n)$ is *on the order of* n^2. Similarly, for a function $f(n) = 3n + 4$, we can say that $f(n) \in o(n^2)$ by proving the following limit: $\lim_{n\to\infty} \frac{3n+4}{n^2} = 0$.

In order to facilitate the reasoning of asymptotic notations in terms of limits, we prove the following two generic theorems.

Theorem 5:
```
⊢ ∀ f  g. (∀ n. 0 < f n ∧ 0 < g n)
  ⇒ (lim(λ n. f n / g n) = 1)
  ⟹ f ∈ Bigo g
```

Theorem 6:
```
⊢ ∀ f  g. (∀ n. 0 < f n ∧ 0 < g n)
  ⇒ (lim(λ n. f n / g n) = 0)
  ⟹ f ∈ Littleo g
```

The proof script of Theorem 5 and 6 is mainly based on the classical definition of sequential limits (given below) and providing the suitable existential variables.

$$\forall x\ x0.\ x \longrightarrow x0 \Leftrightarrow$$
$$\forall e.\ 0 < e \Rightarrow \exists N.\forall n.n \geq N \Rightarrow |(x\ n\ \ x0)| < e$$

The formalization reported in this paper is available for download at [16]. The most important part in the verification process was to pick the right values for the existentially quantified variables as has been described in the proof sketches of some of the theorems.

V. Conclusions

In this paper, we presented a higher-order-logic formalization of asymptotic notations. First of all, we formalized the definitions of O, Θ, Ω, o and ω notations. Then by using these definitions, we formally verified their properties such as transitivity, symmetry, transpose symmetry and reflexivity using the HOL4 theorem prover. The reported formalization facilitates the process of formally analyzing the complexities of algorithms using these notations in HOL4.

An interesting application domain of our formalization is cryptography where asymptotic notations are used to estimate the size of the key so that it will be infeasible to break a system using given number of steps. Similarly, asymptotic notations are frequently used in security assessment of authentication protocols, such as, the security proof of password authentication protocols and the reported formalization can play a vital role in this kind of security-critical analysis.

VI. Acknowledgements

This work is supported by ICT Fund UAE, fund number 21N206 at UAE University, Al Ain, United Arab Emirates.

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 4th ed., 2003.

[2] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill, 2nd ed., 2001.

[3] P. G. H. Bachmann, *Analytische Zahlentheorie, Bd 2: Die Analysische Zahlentheorie*. Teubner, Leipzig, Germany, 1894.

[4] E. Landau, *Handbuch Der Lehre Von Der Verteilung Der Primzahlen*. Teubner, Leipzig, Germany, 1909.

[5] D. E. Knuth, *Big Omicron and Big Omega and Big Theta*. ACM SIGACT News, 8:1824, 1976 Teubner, Leipzig, Germany, 1909.

[6] J. E. Hopcroft and R. E. Tarjan, "Dividing a Graph into Triconnected Components," *SIAM Journal of Computing*, vol. 2, no. 3, pp. 135–158, 1973.

[7] C. Gutwenger and P. Mutzel, "A Linear Time Implementation of SPQR-Trees," in *Graph Drawing*, vol. 1984 of *Lecture Notes in Computer Science*, pp. 77–90, Springer, 2001.

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

[9] R. Krueger, P. Rudnicki, and P. Shelley, "Asymptotic notation. part i: Theory 1," *Journal of Formalized Mathematics*, vol. 11, pp. 1–7, 2003.

[10] J. Avigad and K. Donnelly, "Formalizing O Notation in Isabelle/HOL," in *Automated Reasoning*, vol. 3097 of *Lecture Notes in Computer Science*, pp. 357–371, Springer, 2004.

[11] M. Eberl, "Proving Divide and Conquer Complexities in Isabelle/HOL," *Journal of Automated Reasoning*, vol. 58, no. 4, pp. 483–508, 2017.

[12] K. Slind and M. Norrish, "A brief overview of hol4," in *Theorem Proving in Higher Order Logics*, vol. 5170 of *Lecture Notes in Computer Science*, pp. 28–32, Springer Berlin / Heidelberg, 2008.

[13] T. Mhamdi, O. Hasan, and S. Tahar, "Formalization of Entropy Measures in HOL," in *Interactive Theorem Proving (ITP)*, vol. 6898 of *Lecture Notes in Computer Science*, 2011.

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

[15] R. Milner, "A Theory of Type Polymorphism in Programming," *Journal of Computer and System Sciences*, vol. 17, pp. 348–375, 1977.

[16] http://save.seecs.nust.edu.pk/projects/an, 2018.