Sigma J Eng & Nat Sci 10 (2), 2019, 237-243

Publications Prepared for the Sigma Journal of Engineering and Natural Sciences
Publications Prepared for the ICOMAA 2019 - International Conference on Mathematical Advances and Applications
Special Issue was published by reviewing extended papers

Research Article

INTRODUCTION TO HOL4 THEOREM PROVER

Kübra AKSOY*[1], Sofiène TAHAR[2], Yusuf ZEREN[3]

[1]*Yıldız Technical University, Department of Mathematics, ISTANBUL;* ORCID:0000-0002-4369-3834
[2]*Concordia University, Department of Electrical and Computer Engineering, Montreal-CANADA;* ORCID:0000-0002-5537-104X
[3]*Yıldız Technical University, Department of Mathematics, ISTANBUL;* ORCID:0000-0001-8346-2208

Received: 21.09.2019 Revised: 21.10.2019 Accepted: 11.11.2019

ABSTRACT

The HOL4 interactive theorem prover is a proof assistant based on Higher-Order Logic. It is an ML language based programming environment in which mathematical functions and predicates can be defined and theorems can be proven. The core of the HOL4 theorem prover is composed of a small set of axioms and inference rules, making proofs in HOL4 sound and trustable. The HOL4 prover includes several theories (libraries) that cover most subjects of classical mathematics. The tool also provides a set of built-in decision procedures that can help automatically prove many simple theorems of arithmetic and Boolean algebra. In this paper we provide an introduction to the HOL theorem prover and show how this tool can be used in the formal analysis of advanced mathematics problems.

Keywords: Formal methods, theorem proving, higher-order logic, HOL theorem prover, HOL4.

1. INTRODUCTION

Formal methods construct a computer based mathematical model of a system and its specification [1], which are then used for mathematical reasoning to check functional properties of interest. That is, formal methods are used as a computer-based tool to mathematically analyze the properties of a system. Similarly, simulation is used traditionally for the analysis of systems. It has, however, some limitations like the exponential explosion of the test cases or the usage of numerical approximation, hence, simulation provides less accurate results. On the other hand, using formal methods, we obtain accurate results because we consider all cases implicitly. Nevertheless, sometimes it is limited and time consuming.

Most widely used formal methods are Model Checking and Theorem Proving [1]. The former is a state-based technique. It is used to verify temporal properties exhaustively over the entire state-space. Model Checking is automated because it is based on propositional logic. Theorem Proving is a proof system which includes types, axioms and inference rules. It allows using logical reasoning to verify relationships interactively based on propositional, first or higher-order logic.

[*] Corresponding Author: e-mail: kubraaksoy22@gmail.com, tel: (212) 383 43 11

Higher-order logic [2] gives simple formalisms with precise semantics. It allows the use of many familiar mathematical notations and suffices for the development of much of classical mathematics. That is, we can formalize notions and develop mathematical structures using higher order logic for verifying properties and proving theorems.

The HOL theorem prover [3] is an interactive proof environment that supports mathematical reasoning on different theories and their applications. It has been developed by Mike Gordon at the University of Cambridge for conducting proofs in higher-order logic. His works show both hardware verification or traditional logical formalizations and how a simple programming language could be semantically embedded in higher-order logic [4]. Over the years, the range of applications of the HOL theorem prover has significantly expanded. The most recent version of the HOL proof assistant, HOL4 [5], is now used for mechanized theorem proving in many areas, including formalization of pure mathematics, design and verification of critical and real-time systems, program refinement, program correctness, compiler verification and concurrency. In this paper, we use this proof tool for illustration purposes.

In this paper, we provide a brief introduction to HOL theorem proving and illustrate how this tool can be utilized in the formal analysis of advanced mathematics issues. The rest of the paper is organized as follows: Section 2 presents some preliminaries about the theorem proving and logics that will facilitate the understanding of the rest of the work. Section 3 describes the HOL4 theorem prover. The proofs in HOL4 and an application are given in Section 4. Finally, we conclude the paper in Section 6.

2. PRELIMINARIES

2.1. Theorem Proving

Theorem proving is a field of computer science and mathematical logic that allows to conduct computer-assisted formal proofs of the correctness of systems using mathematical reasoning. It is one of the most used formal methods to verify a system and its desired properties in appropriate logic. While a system is modeled as a function, its properties are modeled as theorems in the same logic. This logic can be propositional logic, first-order logic or higher-order logic. Theorems are interactively verified based on mathematical reasoning in a proof system in order to describe the system. For doing this, the implementation and specification of a system are both expressed in terms of logical formulas and the proof of correctness is derived from a very small set of axioms and inference rules. In brief, theorem proving allows us to establish a mathematical proof that the properties are basically satisfied.

There are some advantages and disadvantages of theorem proving. One of the most powerful advantage is being high expressiveness because it is based on higher-order logic which provides expressive notation and high abstraction. We can formalize and verify properties including the underlying theory and assumptions, rather than isolated properties. Morever, the theorem prover guaranteed soundness so that it has less risks of mistakes because each theorem is derived from either previous theorems or the basic axioms. However, because of the use of higher-order logic, only some parts of the proofs can be automated, while the details and proof strategies need explicit human guidance. It also has limited mathematical libraries.

The most widely used theorem provers are HOL4 [5], HOL Light [6], Coq [7], Isabelle/HOL [8], ACL2 [9], PVS [10]. For example, ACL2 is based on first order logic and developed at University of Texas in Austin. PVS and HOL are based on higher-order logic. The former is developed at Stanford Research Institute, and the latter is developed at University of Cambridge.

2.2. Logics

Logic is the science of formal principles of reasoning or correct inference [2]. In a theorem prover, logic is defined by a formal system called proof system. There are many logics such as propositional, predicate and higher-order logic. Propositional logic is reasoning about complete sentence. That is, it is a statement that is either true (T) or false (F). For instance, statements or propositions can be ‘Elephants fly’, ‘Milk is white’ and ‘5 < 8’, and according to our present knowledge, the first is false and the last two are true. This logic has Boolean operators such as and (∧), or (∨), if. . . then (⇒), if and only if (⇔), not (¬). Combinational logic and finite-state transition systems can be modelled using Boolean formulas and variables. However, for modelling of complex systems this logic cannot be enough.

First-order logic, known as predicate logic, is used to express individual objects and relationship between them. It consists of constants, variables and predicates. In a universe of discourse, constants are represented as specific objects, variables range over objects, and predicates use properties of objects or relationship between objects. Moreover, predicates are often associated with sets. We can quantify over variables using the universal quantifier (∀) which refers to all object and the existential quantifier (∃) which refers to for some object. Quantification over relations greatly enhances the expressive power of first order formulas. For example,

$$\forall x \exists y \, ((P(x) \vee \neg Q(y)) \rightarrow (Q(x) \rightarrow P(y)))$$

Higher-order logic is a form of predicate logic in which quantification is used over arbitrary predicates and functions. Variables can be functions and predicates. Functions and predicates can take functions as arguments and return functions as values. Besides, predicates in higher-order logic may be interpreted as sets of sets. Higher-order logic is different from first-order logic by means of the addition of variables for subsets, relations and functions of the universe. So, it is highly expressive and can be used to describe any mathematical relationship. For instance,

$$\forall xy. \, \exists P. \, P \; xy$$

Figure 1 provides a comparison between logics based on certain characteristics which are expressiveness, decidability and completeness. The meanings of these as follows [4]:

- Expressiveness is the capability to describe complex mathematical models.
- Decidability means there is an algorithm for deciding the (sementical) truth of any formula (theorem).
- Completeness means all valid formulas that are semantically true are provable

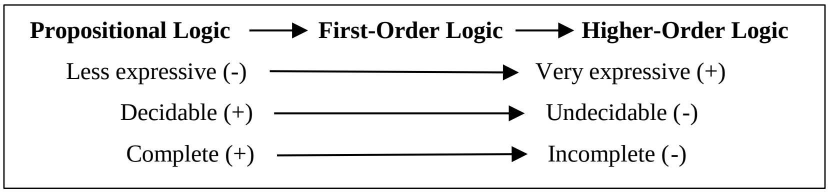

Figure 1. Comparison between Logics

Propositional logic with truth tables is decidable and complete; however, because of the logic itself it is not so expressive. On the other hand, higher-order logic is very expressive because it allows quantifications, and reasoning about all kinds of mathematics such as real numbers, integral, set theory, etc. It is neither complete nor decidable.

3. HOL4 THEOREM PROVER

The HOL4 interactive theorem prover is a proof assistant for higher-order logic providing a programming environment in which theorems can be proved and proof tools implemented [11]. HOL4 consists of a notation (syntax), a small set of five fundamental axioms (facts) and a small set of eight inference (deduction) rules. Using higher-order logic in it, we can translate input from concrete syntax to abstract syntax, and translate output back from abstract to concrete syntax. HOL4 has its own notation. Table 1 provides the mathematical interpretations of some frequently used HOL notation.

Table 1. HOL Notation

Standard Symbol	HOL Symbol	Description
¬	~	Logical negation
∨	\/	Logical or
∧	/\	Logical and
⇒	=>	Implication
⇔	<=>	Equivalence
≠	< >	Disequiation
∀x. t	!x. t	For all x: t
∃x. t	?x. t	For some x: t
$\lambda x. f$	$\lambda x. f$	Function that maps x to f(x)
εx. t(x)	εx. t(x)	Some x such that t(x) is true

A HOL theory is collection of types, lemmas, functions and tactics. HOL theories are databases of already proved theorems. The HOL theorem prover has a very rich collection of libraries such as Boolean, numbers, integers, real, rational, probability, integration, etc. For example, in the arithmetic theory, the main type is num and the function SUC n means n+1. It contains relational operators (<, >, ≤ , = , etc.) as well as arithmetic operators (+, -, ×, ÷, etc.). This theory also has many other useful functions such as max, min, odd, even.

Soundness is assured because every new theorem must be created from either the basic axioms and primitive inference rules or any other already proved theorems. HOL4 is open source and supports propositional, predicate and higher-order logic. It provides formal verification frameworks for both software and hardware. It is also a platform for the formalization of pure mathematics. All theorems in HOL ultimately proved using only the basic axioms and primitive inference rules. Some axioms are in Figure 2 and some inference rules are given in Figure 3:

BOOL_CASES :	ETA:	SELECT :
$\vdash (\forall t. t \vee \neg t)$	$\vdash (\lambda x.\ M\ x) = M$	$\vdash \forall\ P\ x. P\ x \implies P\ (\varepsilon\ y.\ P\ y)$

Figure 2. Some HOL Axioms

ASSUME :	REFL :	BETA CONVERSION :
$\{t\} \vdash t$	$\vdash t = t$	$\vdash (\lambda x.\ t)v = t[v/x\,]$
TRANS :	ABSTRACTION :	COMB:
$\dfrac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}$	$\dfrac{\Gamma \vdash s = t \quad \text{x not free in } \Gamma}{\Gamma \vdash \lambda x. s = \lambda x. t}$	$\dfrac{\Gamma \vdash s = t \quad \Delta \vdash s = t\ types\ fit}{\Gamma \cup \Delta \vdash s(u) = t(v)}$

Figure 3. Some HOL Inference Rules

In Figures 2 and 3, ⊢ is an infix data-type constructor for the type. The constructor maps a list of terms. Generally, given Γ ⊢ Δ, Γ is the assumption and Δ represents the conclusion. For example, in the Abstraction inference rule, Γ is the assumption and the right-hand side s = t represents the conclusion. The lambda abstraction (λx. fx) represents any function definition and ε is used for Hilbert choice operator.

4. PROOFS IN HOL4 THEOREM PROVER

The logic in HOL system is represented in the strongly-typed functional programming language Meta Language (ML) [12]. ML allows the interaction with the theorem prover to represent higher-order logic theorems using abstract data types. There are mainly two ways to prove theorem, either forward and backward proofs. The main porpose of forward proofs is that axioms and inference rules are used to derive theorems. That is, a forward proof is a way to rewrite the assumptions to reach the proof goal. On the other hand, backward proof means that we rewrite the goal to reach the assumptions. Users generally lay emphasis on backward proofs in HOL4. Backward proofs are implemented by tactics, which are ML functions that break goals into simple subgoals in HOL such as GEN_TAC, EQ_TAC, STRIP_TAC, REWRITE_TAC, etc.

In addition, the combination of forward and backward proof styles is also allowed. 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.

4.1. Example of a Goal and Proof

As an example of backward proof, let us prove Equation (1) in the arithmetic theory.

$$\forall n. n + 1 + 1 = n + 2,\ \ n \in \mathbb{N} \tag{1}$$

Using the HOL theorem prover, if we want to verify the above statement is mathematically correctly interpreted. To start, we should write this statement as a goal via the ML function “g” , using HOL notation.

$$g'!\,(n : num). n + 1 + 1 = n + 2\,\grave{}\,;$$

A goal is verified in HOL based on already existing theorems and definitions. Each simplification step is applied using the ML function “e”. The first simplification step is usually to remove the forall-quantifiers because it can be easily added later. This can be done using **GEN_TAC** as follows:

```
e (GEN_TAC);
```

In order to find a theorem in a certain theory we use **DB.match["theory"]**. Hence, we specify the expression of the term we are interested in as follows:

$DB.match$ ["arithmetic"] ($Term$ `$(a : num) + (b + c)$`) ;

This returns the Addition Associativity theorem **ADD_ASSOC :**

$$\vdash \forall\, m\, n\, p.\ m + (n + p) = m + n + p$$

Now, we want to simplify our proof goal using the symmetry of the Addition Associativity via **GSYM ADD_ASSOC** and then we rewrite the goal using **REWRITE_TAC[]**. Combined together we apply :

e (REWRITE_TAC[GSYM ADD_ASSOC]);

and obtain following:

$$\boxed{\text{subgoal} : \text{n} + (1 + 1) = \text{n} + 2}$$

By this, we achieved the first step of our mathematical reasoning in HOL. In the next step, we want to simplify the left-hand side of the equation. For doing this, we use again **DB.match** in order to find a similar theorem in the Arithmetic theory, as follows :

$DB.match$ ["arithmetic"] ($Term$ `$(n : num) + b = (n + a)$`) ;

and find the theorem named **EQ_ADD_LCANCEL** :

$$\vdash \forall\, m\, n\, p. (m + n = m + p) \Leftrightarrow (n = p)$$

Upon rewriting, as follows:

e (REWRITE_TAC[EQ_ ADD_LCANCEL]);

we obtain the following output:

$$\boxed{\text{subgoal} :\ 1 + 1 = 2}$$

Similarly, we use **DB.match** to simplify further our proof goal as follows:

$DB.match$ ["arithmetic"] ($Term$ `$(2{:}num)$`);

and find the theorem named **TWO** :

$$\vdash 2 = SUC\ 1$$

Now, for the last step of our proof goal we need to fing a theorem related to the function **SUC** as follows:

$DB.match$ ["arithmetic"] ($Term$ `$(SUC)$`);

This theorem is called **ADD1**:

$$\vdash \forall\, m. SUC\ m = m + 1$$

In particular, above theorems can be written as follows :

e (REWRITE_TAC[TWO, ADD1]);

In summary, the goal is proved with the below proof script :

```
GEN_TAC THEN
REWRITE_TAC[GSYM ADD_ASSOC] THEN
REWRITE_TAC[EQ_ADD_LCANCEL] THEN
REWRITE_TAC[TWO,ADD1];
```

and obtain following theorem :

$$\forall\, n. n + 1 + 1 = n + 2,\ \ n \in \mathbb{N}$$

We conducted the above proof in order to show how the HOL theorem proving steps look like in details. However, for such simple arithmetic goal, the HOL4 theorem prover has built-in automated tactics such as **METIS_TAC** and **PROVE_TAC**, which can be used to prove the above goal automatically in one single step.

5. CONCLUSION

In this paper, we provided a brief introduction to the HOL theorem proving. This is the first such paper presented to the mathematics community. HOL theorem proving is used for mathematical reasoning in a certain logic. Unlike model checking, theorem proving based on higher-order logic has high expressiveness. Thus, we verify generic mathematical expressions. The core of the HOL4 theorem prover has only 5 axioms and 8 primitive inference rules. Every new theorem is obtained from the basic axioms and inference rules or from any other already proven theorems so that its soundness is guaranteed. HOL4 is open source and has wide variety of applications. There are many positive features of the HOL theorem prover. For instance, HOL will not allow us to prove anything wrong because it works like a proof checking/assistant. Furthermore, it provides a record (repository) of proof detailed steps, and thus, users can remember their proof steps even after many years. Besides, we can reuse intermediate proof steps (lemmas) in verifying other theorems. HOL4 can be used in all areas of Mathematics and Sciences, and can even help finding errors in published work. For example, the authors of [13] were able to find an error in the distributivity law published in [14].

Acknowledgement

This research was partially supported by TUBITAK 2221 program.

REFERENCES

[1] O. Hasan and S. Tahar., (2015) Formal Verification Methods, *In Encyclopedia of Information Science and Technology, IGI Global*, 7162 – 7170.

[2] J. Benthem, K. Doets., (2001) *Higher-order logic. In: Handbook of Philosophical Logic*, *Springer*, 1: 189 - 243.

[3] M. Gordon and T. Melham., (1993) Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic, *Cambridge University Press*, New York, USA.

[4] J. Harrison., (2009) Handbook of Practical Logic and Automated Reasoning, *Cambridge University Press,* New York, USA.

[5] HOL4. https://hol-theorem-prover.org/, 2019.

[6] HOL Light. http://www.cl.cam.ac.uk/ jrh13/hol-light/, 2019.

[7] Coq. http://coq.inria.fr/, 2019.

[8] Isabelle/HOL. https://isabelle.in.tum.de/, 2019.

[9] ACL2. http://www.cs.utexas.edu/users/moore/acl2/, 2019.

[10] PVS. http://pvs.csl.sri.com/, 2019.

[11] K. Slind and M. Norrish., (2008) A Brief Overview of HOL4. In: Theorem Proving in Higher Order Logics, *ser. LNCS, Springer*, 5170: 28–32.

[12] L. Paulson., (1996) ML for the Working Programmer, *Cambridge University Press*, New York, USA.

[13] Y. Elderhalli, O. Hasan, and S. Tahar., (2019) A Methodology for the Formal Verification of Dynamic Fault Trees Using HOL Theorem Proving, *IEEE Access*, 7, 1: 136176-136192.

[14] J. Ni, W. Tang and Y. Xing., (2013) A Simple Algebra for Fault Tree Analysis of Static and Dynamic Systems, *IEEE Transactions on Reliability*, 62, 4: 846–861.