



**HAL**  
open science

## Reverse Engineering of HOL Proofs

Shuai Wang

► **To cite this version:**

| Shuai Wang. Reverse Engineering of HOL Proofs. INRIA Paris-Rocquencourt. 2015. hal-04442046

**HAL Id: hal-04442046**

**<https://hal.science/hal-04442046>**

Submitted on 6 Feb 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Reverse Engineering of HOL Proofs

Shuai Wang

July 2015 (revisited in Sep 2016)

# Project Proposal

Thesis submitted in accordance with the requirements of the ENS Cachan and INRIA as MPRI M1 internship report.

- Course: MPRI
- Level: M1
- Personal Tutor: Dr. David Baelde (ENS Cachan)
- Supervisor: Dr. Gilles Dowek
- Institute: INRIA
- Team: Deducteam
- Address: INRIA, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France.

This internship is to analysis the translation of HOL-proofs to Dedukti in order to build constructive proofs when possible. The goal of this project is a first quantitative analysis of the proportion of proofs in usual classical mathematics that do not genuinely use the excluded middle and that could be expressed constructively.

## Declaration Of Authorship

I declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

Signed \_\_\_\_\_ Date \_\_\_\_\_

# Contents

Contents	iii
List of Figures	v
Acknowledgement	vi
Datasets, Code, and Reproducibility	vii
Nomenclature	vii
<b>1 Introduction</b>	<b>1</b>
<b>2 Basics</b>	<b>2</b>
2.1 Higher Order Logic . . . . .	2
2.2 HOL Light . . . . .	4
2.2.1 Introduction to HOL Light . . . . .	4
2.2.2 Types and Terms . . . . .	4
2.2.3 Constants and Connectives . . . . .	4
2.2.4 Inference Rules of HOL Light . . . . .	6
<b>3 HOL Proof Checking with OpenTheory, Holide and Dedukti</b>	<b>8</b>
3.1 OpenTheory and OpenTheory HOL Light . . . . .	8
3.1.1 OpenTheory Version 6 . . . . .	9
3.2 $\lambda\Pi$ -calculus and $\lambda\Pi$ -calculus Modulo and Dedukti . . . . .	10
3.3 Holide . . . . .	12
3.3.1 Translation . . . . .	13
3.3.2 Correctness . . . . .	15
3.3.3 Holide2 . . . . .	15
<b>4 Reverse Engineering of HOL Proofs</b>	<b>17</b>
4.1 Double Negation Translation . . . . .	18
4.2 Kernel Hacking for Reverse Engineering of HOL Proofs . . . . .	19

<b>5</b>	<b>HOLALA</b>	<b>20</b>
5.1	Kernel Hacking . . . . .	20
5.1.1	Logic Kernel . . . . .	20
5.1.2	Proof Logging . . . . .	22
5.2	Holide2x . . . . .	22
5.2.1	HOL Terms . . . . .	23
5.2.2	HOL Proofs . . . . .	23
<b>6</b>	<b>Proof Analysis and Evaluation</b>	<b>24</b>
6.1	Workflow . . . . .	24
6.2	Proof Checking . . . . .	25
6.3	Proof Analysis . . . . .	27
6.4	ProofCloud . . . . .	28
<b>7</b>	<b>Conclusion and Future Work</b>	<b>33</b>
7.1	Conclusion and Contribution . . . . .	33
7.2	Optimisation of proofs . . . . .	33
7.3	HOL-Modulo . . . . .	34
7.4	HOL-Tableaux . . . . .	34
7.5	ProofCloud . . . . .	35
<b>A</b>	<b>Comparison of Sequent Calculus, Natural Deduction and HOL Light</b>	<b>36</b>
<b>B</b>	<b>Proof of Law of Excluded Middle</b>	<b>40</b>
B.1	Law of Excluded Middle . . . . .	40
B.2	Diaconescu’s Proof of Law of Excluded Middle . . . . .	40
B.3	A New Proof of Law of Excluded Middle . . . . .	41
B.4	Proof of $\neg\neg$ -LEM . . . . .	42
<b>C</b>	<b>Attempts of reverse engineering of HOL Light Proofs</b>	<b>43</b>
C.1	HOL-intermediate . . . . .	44
C.2	HOLIU . . . . .	44
C.3	HOLbot . . . . .	45
<b>D</b>	<b>The Specification of ProofCloud</b>	<b>46</b>
D.1	The Specification of ProofCloud . . . . .	46
	<b>Index</b>	<b>47</b>

# List of Tables

2.1	Primitive and Axiomatic Definitions of Connectives and Constants of $Q_0$ . . .	3
2.2	HOL Light Notations . . . . .	5
2.3	Primitive Inference Rules of HOL Light [20] . . . . .	6
2.4	HOL Light Deduced Inference Rules . . . . .	7
3.1	OpenTheory version 5 Logic Kernel [21] . . . . .	9
3.2	New and Updated Inference Rules of OpenTheory (version 6) . . . . .	9
3.3	Differences between version 5 and version 6 . . . . .	10
3.4	Typing Rules of the $\lambda\Pi$ -calculus . . . . .	12
4.1	Properties of Translation Compared . . . . .	19
5.1	Primitive and Axiomatic Definitions of Connectives and Constants Comparison	20
5.2	Additional Primitive Inference Rules of HOLALA . . . . .	22
5.3	Size of Deductive Rules Between OpenTheory HOL Light and HOLALA . . .	22
6.1	Version Correspondance of OpenTheory, Holide and HOLALA . . . . .	25
6.2	Size of Article Files and Translation Time . . . . .	26
6.3	Size of Dedukti Files and Proof Checking Time . . . . .	26
6.4	Size and Frequency Analysis of Main Inference Rules of HOL Light and HO- LALA Proofs . . . . .	28
6.5	Comparison of Translation and Proof Checking . . . . .	30
A.1	Example Proof of $(p \wedge (p \rightarrow q)) \rightarrow q$ in Sequent Calculus, Natural Deduction and HOL Light . . . . .	36
A.2	Inference System Comparison . . . . .	39

# List of Figures

5.1	Dependency of Constants and Connectives Analysis . . . . .	21
6.1	Workflow of HOLALA, Holide, OpenTheory and ProofCloud . . . . .	24
6.2	Frequency of Main Inference Rules of OpenTheory Articles . . . . .	29
6.3	Frequency of Main Inference Rules of HOLALA Articles . . . . .	29
6.4	The Interface of ProofCloud . . . . .	30
6.5	Dependency of Packages of OpenTheory . . . . .	31
6.6	A Proof Page of ProofCloud . . . . .	32
6.7	A Proof Checking Page of ProofCloud . . . . .	32
C.1	Constants and Connectives Dependency Analysis and Design . . . . .	44

# Acknowledgement

The author wishes to thank Dr. Gilles Dowek, Mr. Ali Assaf, Mr. Frédéric Gilbert and Mr. Raphaël Cauderlier, members of Deducteam at INRIA and Dr. Benno van den Berg (ILLC, UvA) for their help during and after this project. The author would also like to thank all those who gave advice or comments during the past few months, especially those at the conferences and workshops (ESLLI, UITP, AITP): Dr. Sven Linker, Mr. Eric Flaten, Mr. Letchi Akhmatov, Dr. Cezary Kaliszyk and anonymous reviewers.

This is a revisited version of the original report with many mistakes corrected. This report also includes a detailed description of the latest version of ProofCloud, which was a side project commered around end of the internship. ProofCloud was later developed as a new platform after the author left INRIA.

Finally, the reference list was reviewed.



# Datasets, Code, and Reproducibility

For reproducibility, the following datasets and their code and documentation are available. Unfortunately. They have not been maintained since early 2016.

1. Holide: The description is on GitHub <https://airobert.github.io/holide/>. The source code is in the repository <https://github.com/airobert/holide>.
2. HOLALA: Some description is on this GitHub page: <https://airobert.github.io/holala/>. The source code is available at <https://github.com/airobert/holala>.
3. ProofCloud: The main website is at <https://airobert.github.io/proofcloud/> and the source code is hosted on GitHub at <https://github.com/airobert/proofcloud>.

The code, intermediate results, design of the interface are all free for use under the licence CC-BY 4.0 provided that the author receives credits via citation or acknowledgement.

# Chapter 1

## Introduction

In Logical Framework, proofs are defined using definitions of logical constants, connectives, quantifiers and axioms. Using a Logical Framework, the theories can be defined explicitly, making it possible to investigate which part of the theory is needed for each given proof. The process of reconstructing proofs in a given form is defined “reverse engineering of proofs”. This work is a first attempt to do such reverse engineering of higher order proofs. More specifically, this project first analyse which proofs in the HOL-light library are constructive. If so, we attempt to transform them in the weaker constructive system. This internship project aimed at such automatic analysis for proofs by picking constructive-compatible definitions and transform proofs using those definitions. As a consequence, rework all the chain from HOL-Light to Dedukti takes these modifications into account for proof checking. Several unsuccessful attempts have been made to change the kernel of HOL Light. Among them HOLALA extends the kernel of HOL Light by introducing the universal quantification and implication. It can classify proofs and perform structural and statistical analysis. The result is that 531 proofs out of 1199 (44%) in the HOL-light standard library are shown to be constructive. The evaluation results showed that HOLALA is a successful attempt to reduce the size of proof files, leading to an improvement of 41.81% in translation time and a speed-up of 38.04% in proof checking. Finally, all structural analysis and proof checking results were presented in a proof search engine, namely ProofCloud.

This report is organised as follows: Chapter 2 and Chapter 3 are an exposition of the tools used during this internship: HOL-Light, Dedukti and Holidé. To perform double negation translation as described in Chapter 4, it is required to do some modification of the primitive symbols of OpenTheory HOL Light kernel. Chapter 5 present a modification of the OpenTheory HOL-Light system, namely HOLALA, which meant to change the definition of the connectives and quantifiers. Following that was the corresponding modifications of Holidé. Chapter 6 presents some proof analysis results and the evaluation.

# Chapter 2

## Basics

### 2.1 Higher Order Logic

Higher Order Logic, also known as simple type theory (STT), is an extension of first order logic [13]. A term is considered either a variable (e.g.  $x$ ), an abstraction (e.g.  $\lambda x.x$ ) or an application (e.g.  $(\lambda x.x) y$ ) of certain type. A variable following  $\lambda$  is considered an argument of a function. Such variables are also considered as *bound*, otherwise *free*. The rest of the function is usually referred to as the *body*. The notation  $x : \iota$  means that the term  $x$  is of type  $\iota$ . The type of a term is recursively defined as  $\iota$  (a type symbol, denoting the type of individuals) or  $\iota \rightarrow \iota$  (a function type, the arrow is known as type constructor), or a boolean type (*bool* for short). Propositions are of type *bool*. Type notions are sometimes omitted for simplicity of representation. Restricted by typing, application has to agree also on their types. For example,  $\lambda x : \iota.x$  which is of type  $\iota \rightarrow \iota$  can be applied to  $w : \iota$  but not  $z : \iota \rightarrow \iota$ . Such functions can be  $\beta$ -reduced through application to another term. Variables bound by  $\lambda$  are considered only place-holders so terms are considered identical if they only differ for the names of their bound variables. For example  $(\lambda x.yx)$  and  $(\lambda s.ys)$ . These terms are considered  $\alpha$ -convertible.  $\beta$ -reduction is simply to replace all the occurrences of the bound variable by the applied term in the “bod” of the function. For example,  $(\lambda x.x)y$  would be  $\beta$ -reduced to  $y$  (denoted as  $(\lambda x.x)y \rightarrow_{\beta} y$ ). The notation  $[y/x]A$  is used to represent the result of replacing all the  $x$  by  $y$  in  $A$ . Thus,  $(\lambda x.yx)z \rightarrow_{\beta} [z/x](yx) = yz$ . The relation  $=_{\beta}$  is the reflexive, symmetric, transitive closure of  $\rightarrow_{\beta}$ , which is also referred to as equivalence up to  $\beta$ . For example,  $(\lambda x.(\lambda y.y)x)w =_{\beta} (\lambda y.y)w =_{\beta} w$ .  $\lambda$ -calculus can be used as representation of arithmetic, lists and more.

Higher order logic includes also some additional axioms and inference rules. Since most mathematical theories can be expressed in STT, it has been well investigated during the past few decades. As a result, automatic and interactive theorem provers have been developed for formal mathematics. Among the interactive theorem provers, the HOL family consists of several interactive theorem provers including HOL4 [33], HOL Light [20], ProofPower [29], HOL Zero [1], etc. Alongside, many higher order automatic theorem provers have also been developed [9, 28, 34].

The following is a brief introduction to the higher order logic system  $Q_0$  [2].  $Q_0$  takes two primitive symbols,  $=$  and  $\iota$ . Other constants and connectives are defined based on it. Table 2.1 illustrates a list of constants and connectives defined in  $Q_0$ . Note that  $\forall x.A$  is an abbreviation of  $\forall(\lambda x.A)$ , which is also written as  $\forall A$  in some contexts.

Symbol	Definition
$=$	primitive
$\iota$	primitive
$\leftrightarrow$	$=$
$\top$	$= = =$
$\forall$	$\lambda p.A = \lambda p.\top$
$\perp$	$\lambda x.T = \lambda x.x$
$\wedge$	$\lambda x\lambda y(\lambda g.g\top\top = \lambda g.gxy)$
$\rightarrow$	$\lambda pq.(p = (p \wedge q))$
$\exists$	$\neg\forall x\neg A$
$\vee$	$\lambda xy.\neg(\neg x \wedge \neg y)$
$\neg$	$\lambda x.x = \perp$
$\exists!$	$(\exists p) \wedge (\forall xy.px \wedge py \Rightarrow x = y)$

Table 2.1: Primitive and Axiomatic Definitions of Connectives and Constants of  $Q_0$

The higher order logic inference system  $Q_0$  takes a list of axioms but has a single primitive inference rule, Rule R. The following illustrates a list of axioms of  $Q_0$  followed by its only inference rule [2].

**Axiom 1.**  $f\top \wedge f\perp = \forall x.fx$

**Axiom 2.**  $(x = y) = (fx = fy)$

**Axiom 3.**  $(f = g) = \forall x.(fx = gx)$

**Axiom 4.**  $(\lambda x.B)A = B$ , where  $B$  is a primitive constant or variable distinct from  $x$ .

**Axiom 5.**  $(\lambda x.x)A = A$

**Axiom 6.**  $(\lambda x.(BC)A = ((\lambda x.B)A)(\lambda x.C)A)$

**Axiom 7.**  $(\lambda x.(\lambda y.B))A = (\lambda y.((\lambda x.B)A))$ , where  $y$  is distinct from  $x$  and from all variables in  $A$ .

**Axiom 8.**  $(\lambda x.(\lambda x.B))A = \lambda x.B$

**Axiom 9.**  $\iota(= y) = y$

**Rule R.**  $\frac{C \quad A = B}{C'} R$ , where  $C'$  is  $C$  with one occurrence of  $A$  replaced by  $B$ , provided that the occurrence of  $A$  in  $C$  is not (an occurrence of a variable) immediately preceded by  $\lambda$ . [2].

To construct proofs, it is first required to understand the definition of a sequent. A sequent is in the form  $\Gamma \vdash A$ , where  $\Gamma$  is the hypotheses and  $A$  the conclusion. Such sequent means that we can deduce  $A$  from  $\Gamma$ . Informally, a proof is a tree of sequents connected by inference rules. Based on the logic of  $Q_0$ , we can deduce some more inference rules corresponding to common logic connectives and constants, making it a usable inference system. One of the most used inference systems is Natural Deduction. It is commonly used in proof assistants including HOL Light.

## 2.2 HOL Light

### 2.2.1 Introduction to HOL Light

HOL Light is an open source interactive theorem prover and proof checker for higher order logic [20]. HOL Light is a simplified version of the original HOL [33] system with a small and reliable kernel. Similar to the HOL system, HOL Light takes advantage of the powerful functional programming language OCaml [20] and has a special notation system for logical connectives and constants which is presented in Table 2.2 in case of confusion.

### 2.2.2 Types and Terms

To understand how HOL Light works, it is necessary to understand how HOL Light defines types and terms. HOL Light terms are usually represented in backticks. For example, ``x:A` means a variable  $x$  of type  $A$ . Each HOL Light term has a type defined recursively, which can be either a type variable, a boolean type or an arrow type. An arrow type is of the form  $\alpha \rightarrow \beta$ , which means that the function takes an argument of type  $\alpha$  and returns a term of type  $\beta$ . A HOL Light term is a variable, a constant, an application or an lambda abstraction. For example, the type of  `$\lambda x : A. \top$`  is  $A \rightarrow bool$ . The syntax of HOL Light is as follows [5]:

type variables	$\alpha, \beta$
type operators	$p$
types	$A, B ::= \alpha \mid p(A_1, \dots, A_n)$
term variables	$x, y$
term constants	$c$
terms	$M, N ::= x \mid \lambda x : A. M \mid MN \mid c$

### 2.2.3 Constants and Connectives

HOL Light's logic is an extension of Church's simple type theory (STT) [7]. STT, together with polymorphic type variables and some axioms and inference rules, makes the logic of HOL Light. HOL Light has terms of type boolean, individual and function as well as polymorphic

HOL Notation	Meaning	Logic Notation	Definition
=	Equivalence	=	primitive
\	lambda	$\lambda$	built-in symbol
!	Universal Quantifier	$\forall$	$\lambda p.A = \lambda p.\top$
?	Existential Quantifier	$\exists$	$\neg\forall x\neg A$
~	Negation	$\neg$	$\lambda p.p \Rightarrow \perp$
==>	Implication	$\Rightarrow$	$\lambda pq.(p = (p \wedge q))$
<=>	Bi-implication	$\Leftrightarrow$	=
\	Disjunction	$\vee$	$\lambda xy.\neg(\neg x \wedge \neg y)$
\&	Conjunction	$\wedge$	$\lambda x\lambda y(\lambda g.g\top\top = \lambda g.gxy)$
T	Truth	$\top$	$\lambda p.p = \lambda p.p$
F	False	$\perp$	$\lambda x.T = \lambda x.x$
@	Hilbert's Choice Symbol	@	Constant
?!	Exist Unique	$\exists!$	$(\exists p) \wedge (\forall xy.px \wedge py \Rightarrow x = y)$

Table 2.2: HOL Light Notations

type variables. There are constants with polymorphic types: equality (=)<sup>1</sup> and Hilbert's choice symbol (denoted as @) [20]. The choice symbol can be considered a function with a predicate as input, and return an element satisfying such predicate. It returns a random element if there is no such element. The choice symbol is used in the axiom of choice, which is in the center of the proof of the law of excluded middle to be presented in Section B.2.

$$\begin{aligned} =: & \alpha \rightarrow \alpha \rightarrow o \\ @: & (\alpha \rightarrow o) \rightarrow \alpha \end{aligned}$$

Here, the polymorphic type variables  $\alpha$  of = would be instantiated to the type of the term(s) it is applied to. For instance, in the term  $\top = \top$ , = is of type  $o \rightarrow o \rightarrow o^2$ .

The constant = plays three roles in HOL Light:

### Definition

= is the first connective introduced HOL Light system. Its first use is to define other logical constants and connectives, as the role of equivalence ( $\equiv$ ).

### Equality

In most cases, = is used to represent equivalence of two terms, including variables, constants,  $\lambda$  abstraction and application. Terms are of the same type on both sides of =.

### Bi-implication

In HOL Light, the bi-implication connector  $\Leftrightarrow$  is redefined as equality since such case can be regarded as equality with type  $o \rightarrow o \rightarrow o$ .

<sup>1</sup> $s = t$  is a conventional concrete syntax for  $((=)st)$ .

<sup>2</sup>This is also the type of bi-implication. Bi-implication ( $\Leftrightarrow$ ) can be considered a special case of the polymorphic =

## 2.2.4 Inference Rules of HOL Light

The definition of the logical symbols and the primitive inference rules are the very beginning of the HOL-Light library. This is known as the kernel of HOL-Light. More specifically, the kernel defines 10 primitive inference rules as in Table 2.3<sup>3</sup>.

**Primitive Inference Rules of HOL Light**

Structural	$\frac{}{\{A\} \vdash A} \text{ASSUME}$
$\lambda$ Calculus	$\frac{\Gamma \vdash A = B}{\Gamma \vdash \lambda x. A = \lambda x. B} \text{ABS}$ $\frac{}{(\lambda x. A)x = A} \text{BETA}$
Instantiation	$\frac{\Gamma[x_1, \dots, x_n] \vdash A[x_1, \dots, x_n]}{\Gamma[t_1, \dots, t_n] \vdash A[t_1, \dots, t_n]} \text{INST}$ $\frac{\Gamma[\alpha_1, \dots, \alpha_n] \vdash A[\alpha_1, \dots, \alpha_n]}{\Gamma[\gamma_1, \dots, \gamma_n] \vdash A[\gamma_1, \dots, \gamma_n]} \text{INST\_TYPE}$
Bi-implication	$\frac{\Gamma \vdash A = B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B} \text{EQ\_MP}$ $\frac{\Gamma \vdash A \quad \Delta \vdash B}{(\Gamma \setminus \{B\}) \cup \Delta \setminus \{A\} \vdash A = B} \text{DEDUCT\_ANTISYM\_RULE}$
Equality	$\frac{}{\vdash A = A} \text{REFL}$ $\frac{\Gamma \vdash A = B \quad \Delta \vdash C = D}{\Gamma \cup \Delta \vdash A(C) = B(D)} \text{MK\_COMB}$ $\frac{\Gamma \vdash A = B \quad \Delta \vdash B = C}{\Gamma \cup \Delta \vdash A = C} \text{TRANS}$

Table 2.3: Primitive Inference Rules of HOL Light [20]

On the bases of the ten primitive inference rules we can deduce inference rules, referred to as derived inference rules" in contrast with "primitive inference rules" in this report (Table 2.4).

<sup>3</sup>To keep tables small, types are eliminated.

Symbol	HOL Light Deduced Inference Rules
$\top$	$\frac{}{\Gamma \vdash \top} TRUTH$
$\perp$	$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} CONTR$
$\wedge$	$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} CONJUNCT1$
	$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} CONJUNCT2$
	$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \cup \Delta \vdash A \wedge B} CONJ$
$\vee$	$\frac{\Gamma \vdash A \vee B \quad \Gamma_1, A \vdash C \quad \Gamma_2, B \vdash C}{\Gamma \cup (\Gamma_1 \setminus \{A\}) \cup (\Gamma_2 \setminus \{B\}) \vdash C} DISJ\_CASES$
	$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} DISJ1$
	$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} DISJ2$
$\Rightarrow$	$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B} MP$
	$\frac{\Gamma \vdash B}{\Gamma \setminus \{A\} \vdash A \rightarrow B} DISCH$
$\forall$	$\frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \forall x A} GEN$ if x is not free in $\Gamma$
	$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} SPEC$
$\exists$	$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A} EXISTS$
	$\frac{\Gamma \vdash \exists x A[x] \quad \Delta, A \vdash B}{\Gamma \cup \Delta \setminus \{A[v/x]\} \vdash B} CHOOSE$
$\neg$	$\frac{\Gamma \vdash \neg A}{\Gamma \vdash A \rightarrow \perp} NOT\_ELIM$
	$\frac{\Gamma \vdash \neg A = \perp}{\Gamma \vdash \neg A} EQF\_ELIM$
	$\frac{\Gamma \vdash A \rightarrow \perp}{\Gamma \vdash \neg A} NOT\_INTRO$
	$\frac{\Gamma \vdash \neg A}{\Gamma \vdash A = \perp} EQF\_INTRO$

Table 2.4: HOL Light Deduced Inference Rules



## Chapter 3

# HOL Proof Checking with OpenTheory, Holide and Dedukti

Higher order inference systems have played an important role in formal mathematics, software verification and hardware verification. The necessity of proof checking follows from the need of correctness of such systems and rooted in three reasons: 1) proof systems may have bugs and may lead to errors in proofs but not apparent within the proof system themselves. 2) Proofs nowadays can be huge, making impossible to check by hand. For example [18] takes a team of scientists a few years to complete. 3) since Simple Type Theory is not sound and complete [13], there is no guarantee for proofs in proof systems to be error free. Cousineau and Dowek showed that Simple Type Theory can be embedded in the  $\lambda\Pi$ -calculus Modulo as well as other Pure Type System (PTS) [10]. This laid the logic foundation of Dedukti [31], a universal proof checker. On top of Dedukti, Holide [5] was developed to transform proofs from OpenTheory to Dedukti.

### 3.1 OpenTheory and OpenTheory HOL Light

It is necessary to have a certain amount of proofs both for testing, transformation and evaluation purposes. In this project, some small proofs were written for testing and debugging reasons but most proofs were retrieved from the OpenTheory library [21]. Although proofs have been developed in HOL Light [20], HOL4 [33] and ProofPower [29], they each contain significant theory formalizations that are not accessible to each other [5, 21]. HOL Light has a formalization of complex analysis while HOL4 has a formalization of probability theory [20, 21, 33]. In contrast, ProofPower has a formalization of the Z specification language [21, 29]. Taking advantage of the similarity of the logic between these systems, OpenTheory has developed a standard article file format for serializing proofs of higher order logic [21]. Table 3.1 shows the OpenTheory logic kernel which has some small differences compared with the HOL Light kernel<sup>1</sup>. The OpenTheory<sup>2</sup> is a cross-platform proof package

---

<sup>1</sup>Note that AppThm Corresponds to MK\_COMB in HOL Light, not AP\_THM.

<sup>2</sup><http://opentheory.gilith.com/>

manager for proofs in theorem provers of HOL family. OpenTheory is organised into packages by theory. Such theories include lists, natural numbers, functions, etc<sup>3</sup>. OpenTheory comes along with the idea of "theory engineering" and has inspired further exploration on theory management between HOL families [15,23] as well as the development of several projects [5,6] takes advantage of these packages. OpenTheory HOL-Light<sup>4</sup> is a modified version of HOL Light with the ability to export standard theory library in the OpenTheory standard proof format (.art). The standard library of OpenTheory can be considered equivalent to the proofs in OpenTheory HOL Light [21].

$$\begin{array}{c}
\frac{}{\vdash t = t} \text{ refl } t \quad \frac{}{\{\varphi\} \vdash \varphi} \text{ assume } \varphi \quad \frac{\Delta \vdash \varphi \quad \Gamma \vdash \varphi = \psi}{\Gamma \cup \Delta \vdash \psi} \text{ eqMp} \\
\\
\frac{\Gamma \vdash t = u}{\Gamma \vdash (\lambda v. t) = (\lambda v. u)} \text{ absThm } v \quad \frac{\Gamma \vdash f = g \quad \Delta \vdash x = y}{\Gamma \cup \Delta \vdash fx = gy} \text{ appThm} \\
\\
\frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{(\Gamma \setminus \{\psi\}) \cup (\Delta \setminus \{\varphi\}) \vdash \varphi = \psi} \text{ deductAntisym} \quad \frac{\Gamma \vdash \varphi}{\Gamma[\sigma] \vdash \varphi[\sigma]} \text{ subst}\sigma \\
\\
\frac{}{\vdash (\lambda v. t)u = t[u/v]} \text{ betaConv}((\lambda v. t)u) \quad \frac{}{c = t} \text{ defineConst } ct \\
\\
\frac{\vdash \varphi t}{\vdash \text{abs}(\text{rep } a) = a \quad \vdash (\text{rep}(\text{abs } r) = r) = \varphi r} \text{ defineTypeOp } n \text{ abs rep } vs
\end{array}$$

Table 3.1: OpenTheory version 5 Logic Kernel [21]

### 3.1.1 OpenTheory Version 6

OpenTheory expanded the logic kernel with four inference rules in version 6: *proveHyp*, *trans*, *sym* and *defineTypeOp*(as in Table 3.2). This made some software depending on it out of date. A full comparison table between version 5 and version 6 is adapted from the announcement<sup>5</sup>.

$$\begin{array}{c}
\frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma \cup (\Delta \setminus \{\varphi\}) \vdash \psi} \text{ proveHyp} \\
\\
\frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u} \text{ trans} \\
\\
\frac{\Gamma \vdash \varphi = \psi}{\Gamma \vdash \psi = \varphi} \text{ sym} \\
\\
\frac{\vdash \varphi t}{\vdash \text{abs}(\text{rep } a) = a \quad \vdash \lambda r. \varphi r = (\lambda r. (\text{rep}(\text{abs } r) = r))} \text{ defineTypeOp } n \text{ abs rep } vs
\end{array}$$

Table 3.2: New and Updated Inference Rules of OpenTheory (version 6)

<sup>3</sup><http://opentheory.gilith.com/packages/>

<sup>4</sup><http://src.gilith.com/hol-light.html>

<sup>5</sup>The version 6 announcement is here:

<http://www.gilith.com/pipermail/opentheory-users/2014-December/000461.html>

Command	New or Updated	Description
version	new	Specify the version of article files.
proveHyp trans sym	new	These three new commands provide an efficient implementation of three frequently used derived inference rules to produce more compact article files.
defineConstList	new	The command defineConstList prov with ides a more abstract principle for constant definition. This supports recent extensions to HOL4 and ProofPower.
hdTl	new	The command hdTl provides a destructor function for lists (an inverse to the existing cons command). Together with the new command defineConstList, making it possible to store constants in the dictionary.
defineTypeOp	updated	The command defineTypeOp has been changed so that the theorems giving the defining properties of the abstraction and representation functions have no free variables. They were: $\vdash (\text{abs } (\text{rep } a)) = a;$ $\vdash (\text{rep}(\text{abs } r) = r) = \varphi r$ And they are now $\vdash (\lambda a. \text{abs } (\text{rep } a)) = \lambda a. a;$ $\vdash \lambda r. \varphi r = (\lambda r. \text{rep } (\text{abs } r) = r)$
pragma	new	The new command pragma allows a writer to ask for a reader to take some implementation-dependent action, e.g., to produce debug output.

Table 3.3: Differences between version 5 and version 6

### 3.2 $\lambda\Pi$ -calculus and $\lambda\Pi$ -calculus Modulo and Dedukti

$\lambda\Pi$ -calculus is also known as LF. It is a typed  $\lambda$ -calculus with dependent types. The type of functions is written  $\Pi x : A. B$ . When  $x$  does not appear free in  $B$ , it can be denoted as  $A \rightarrow B$ . An example of dependently typed term is the term  $\lambda x : \text{int}. p : \Pi x : \text{int}. Ax$ . This term means that the type of  $p$  depends on the integer  $x$ . Think of  $p$  as a piece of paper of size  $Ax$ . If  $x$  is 4 then the paper  $p$  is of size  $A4$ . If there exist a paper of size  $A4$ , then the type  $A4$  is inhabitable.  $\lambda\Pi$ -calculus is useful since through the Curry-Howard correspondence, it can represent a variety of logics [19]. There are several formalizations of simple type theory in  $\lambda\Pi$ -calculus such as [3], [30], [5] and [32]. The main idea of this approach is "propositions as types" and "proofs as terms". For each proposition  $\phi$  in HOL, we have a type  $\|\phi\|$  in  $\lambda\Pi$ -calculus.  $\phi$  is provable if and only if the type  $\|\phi\|$  is inhabitable. A proof of  $\phi$  is translated as a term of type  $\|\phi\|$ . Thus, the correctness of the proof is equivalent to the well-typedness of the term [5]. A signature contains constants while a context is a list of variable declarations. The syntax of  $\lambda\Pi$ -calculus is as follows, where  $\cdot$  represents empty signature or empty context.:

term   typet	$t = x \mid \text{Type} \mid \text{Kind} \mid \Pi x. t \ t \mid \lambda x. t \ t \mid t \ t$
contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$
signatures	$\Sigma ::= \cdot \mid \Sigma, c : A$

$\lambda\Pi$ -calculus Modulo is  $\lambda\Pi$ -calculus combined with rewriting technique similar to Deduction Modulo. As an extension of  $\lambda\Pi$ -calculus, in  $\lambda\Pi$ -calculus Modulo, a signature contains constants as well as rewriting rules:

$$\Sigma ::= \cdot \mid \Sigma, c : A \mid \Sigma, [\Gamma] M \rightsquigarrow N$$

In the formula above,  $\Sigma, [\Gamma] M \rightsquigarrow N$  means to add the rewriting rule  $M \rightsquigarrow N$  to the signature  $\Sigma$  regarding the context  $\Gamma$ . To keep type checking decidable, the free variables on the right-hand-side must appear on the left-hand-side and the left-hand-side must follow the higher order pattern fragment [26] in rewrite rules [4, 5]. Deduction Modulo takes advantage of rewriting techniques. There are several notations to be introduced. Assuming  $R$  is a set of rewriting rules,  $\rightarrow_R$  represents induced reduction relation while  $\rightarrow_R^+$  means transitive closure. In addition,  $\rightarrow_R^*$  refers to reflexive transitive closure.  $\equiv_R$  is denoted as the reflexive, symmetric, transitive closure [5]. This thesis follows the notation of [5] and uses  $\beta\Sigma$  as the union of the  $\beta$  rule with the rewrite rules of  $\Sigma$ . The reduction relation  $\rightarrow_{\beta\Sigma}$  should be confluent and strongly normalizing. Table 3.4 contains the typing rules for  $\lambda\Pi$ -calculus Modulo [5].

$$\begin{array}{c}
\frac{\Gamma \text{ context} \quad (x : A) \in \Gamma}{\Gamma \vdash x : A} \text{Var} \\
\frac{\Gamma \text{ context} \quad (c : A) \in \Sigma}{\Gamma \vdash c : A} \text{Const} \\
\frac{\Gamma \text{ context}}{\Gamma \vdash \text{Type} : \text{Kind}} \text{Type} \\
\frac{\Gamma \vdash A : \text{Type} \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A. B : s} \text{Prod} \\
\frac{\Gamma \vdash A : \text{Type} \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A. M : \Pi x : A. B} \text{Abs} \\
\frac{\Gamma \vdash M : \Pi x : A. B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : [N/x]B} \text{App} \\
\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \text{Type} \quad A \equiv_{\beta\Sigma} B}{\Gamma \vdash M : B} \text{Conv} \\
\frac{\Sigma \text{ signature}}{\text{.context}} \text{Emptyctx} \\
\frac{\Gamma \vdash A : \text{Type} \quad x \notin \Gamma}{\Gamma, x : A \text{ context}} \text{VarCtx} \\
\frac{}{\text{.signature}} \text{Emptysig} \\
\frac{\Sigma | \cdot \vdash A : s \quad c \notin \Sigma}{\Gamma, c : A \text{ signature}} \text{ConstSig} \\
\frac{\Sigma | \Gamma \vdash M : A \quad \Sigma | \Gamma \vdash N : A}{\Sigma | \Gamma | M \rightsquigarrow N \text{ signature}} \text{RewriteSig}
\end{array}$$

Table 3.4: Typing Rules of the  $\lambda\Pi$ -calculus

Dedukti<sup>6</sup> is a type checker based on  $\lambda\Pi$ -calculus modulo [31]. Being a logical framework, Dedukti allows the definition of different theories, for instance classical/constructive HOL and Calculus of Constructions [31]. With the help of Holide, CoqInE, Focalide and Krajono<sup>7</sup>, it can perform proof checking on Coq, FoCaLize and Matita proofs<sup>8</sup> [6]. It has been evaluated to be an efficient and reliable proof checker.

### 3.3 Holide

Holide is translator from HOL proofs to Dedukti [31] developed by Ali Assaf and Guillaume Burel [5]. It accepts files in the OpenTheory standard proof format (.art) and produces files (.dk) for the Dedukti proof checker [5]. The first version translated OpenTheory standard

<sup>6</sup> Available at <https://www.rocq.inria.fr/deducteam/Dedukti>.

<sup>7</sup> <http://dedukti.gforge.inria.fr/>

<sup>8</sup> <https://who.rocq.inria.fr/Ali.Assaf/research/krajono-deducteam-2015-slides.pdf>

library and performed proof checking [5]. This internship project also includes an upgrade of Holidé following the most recent updates of OpenTheory (version 6). The translation is presented next.

### 3.3.1 Translation

A signature contains constants and rewriting rules while a context is a list of variable declarations. The following are the translation of HOL types, HOL terms and HOL proofs.

**HOL Types** To translate HOL types we need to introduce a Dedukti type: `Type` as well as the follows constructors:

- `type : Type`
- `bool : type`
- `ind : type`
- `arrow : type → type → type`

HOL types could then be translated as Dedukti terms:

- $|\alpha| = \alpha$
- $|\text{bool}| = \text{bool}$
- $|\text{ind}| = \text{ind}$
- $|A \rightarrow B| = \text{arr } |A| |B|$

For an n-ary HOL type operator  $p$ , which is of type  $\underbrace{\text{type} \rightarrow \dots \rightarrow \text{type}}_n \rightarrow \text{type}$ , an instance  $p(A, \dots, A_n)$  is then translated to the term  $p|A_1| \dots |A_n|$ .

**HOL Terms** We define a new dependent type called *term*, which depends on a Dedukti type. `eq` and `select` are also introduced for equality and the choice operator respectively.

- `term : type -> Type`
- `eq: Πα : type.term(arr α(arr α bool))`
- `select : Πα : type.term(arr (arr α bool)α)`

To translate a HOL type  $A$  as a term, we have  $||A|| = \text{term } |A|$ . For any HOL term  $M$ , we have the translation  $|M|$  defined as follows:

- $|x| = x$
- $|M N| = |M| |N|$
- $\lambda x : A.M| = \lambda x : ||A||.|M|$
- $| (=_{A}) | = eq|A|$
- $|select_A| = select|A|$

Holide has also introduced a rewriting rule to ensure terms are well-typed [5].

$$[\alpha : type, \beta : type]term(arr \alpha \beta) \rightsquigarrow term\alpha \rightarrow term\beta$$

In addition, for every family of HOL constant  $c$  of type  $A$  with free variables  $\alpha_1, \dots, \alpha_n$ , we declare a new constant of type  $\Pi\alpha_1 : type. \dots \Pi\alpha_n : type.||A||$ . And an instance of the constant  $c_{A_1, \dots, A_n}$  by  $c|A_1| \dots |A_n|$ .

**HOL Proofs** Similar as terms, we introduce another constant with dependent type:  $proof : term\ bool \rightarrow Type$ , indexed by the corresponding HOL proposition. For any context  $\Gamma = \phi_1 \dots \phi_n$ , we define  $||\Gamma|| = h_{\phi_1} : ||\phi_1||, \dots, h_{\phi_n} : ||\phi_n||$ , where  $||\phi|| = proof\phi$  and  $\phi_1 \dots \phi_n$  are fresh variables [5]. We declare the following constants:

- Refl:  $\Pi\alpha : type. \Pi x : term\ \alpha. proof(eq\ \alpha\ x\ x)$
- FunExt:  $\Pi\alpha, \beta : type. \Pi f, g : term(\text{arrow}\ \alpha\ \beta). \Pi x : term\ \alpha. proof(eq\ \beta(f\ x)(g\ x)) \rightarrow proof(eq(\text{arrow}\ \alpha\ \beta)\ f\ g)$
- AppThm:  $\Pi\alpha, \beta : type. \Pi f, g : term(\text{arrow}\ \alpha\ \beta). \Pi x, y : term\ \alpha. proof(eq(\text{arrow}\ \alpha\ \beta)\ f\ g) \rightarrow proof(eq\ \alpha\ x\ y) \rightarrow proof(eq\ \beta(f\ x)(g\ y))$
- PropExt :  $\Pi p, q : term\ bool. (proof\ q \rightarrow proof\ p) \rightarrow (proof\ q \rightarrow proof\ p) \rightarrow proof(eq\ bool\ p\ q)$
- EqMp :  $\Pi p, q : term\ bool. proof(eq\ bool\ p\ q) \rightarrow proof\ p \rightarrow proof\ q$

The proofs can then be translated as:

The constant FunExt is short for *functional extensionality*<sup>9</sup>, which is used for the translation of both AbsThmas well as the  $\eta$  axiom [5]. Translation of rules are provided as follows, where  $i$  is the index of proof  $D_i$ .

---

<sup>9</sup>Functional extensionality means that if two functions  $f$  and  $g$  are of type  $A \rightarrow B$  and are equal on all values  $x$  of type  $A$ , then  $f$  and  $g$  are equal.

$$\begin{array}{|l}
\frac{}{\vdash t = t} \text{ refl } t \quad \Big| = \text{RefI} |A| |M|, \text{ where } M : A \\
\frac{\Gamma \vdash t = u}{\Gamma \vdash (\lambda v.t) = (\lambda v.u)} \text{ absThm } v \quad \Big| = \text{FunExt} |A| |B| |\lambda x : A.M| |\lambda x : A.N| (\lambda x : |A|. |D|) \\
\frac{\Gamma \vdash f = g \quad \Delta \vdash x = y}{\Gamma \cup \Delta \vdash fx = gy} \text{ appThm} \quad \Big| = \text{AppThm} |A| |B| |F| |G| |M| |N| |D_1| |D_2| \\
\frac{}{\vdash (\lambda v.t)u = t[u/v]} \text{ betaConv}((\lambda v.t)u) \quad \Big| = \text{RefI} |B| |M|, \text{ where } M \text{ is of type } B \\
\frac{}{\{\varphi\} \vdash \varphi} \text{ assume } \varphi \quad \Big| = h_\varphi, \text{ where } h_\varphi \text{ is a fresh variable} \\
\frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{(\Gamma \setminus \{\psi\}) \cup (\Delta \setminus \{\varphi\}) \vdash \varphi = \psi} \text{ deductAntisym} \quad \Big| = \text{ProfExt} |\phi| |\psi| (\lambda h_\psi : ||\psi||. |D_1|) (\lambda h_\phi : ||\phi||. |D_2|) \\
\frac{\Delta \vdash \varphi \quad \Gamma \vdash \varphi = \psi}{\Gamma \cup \Delta \vdash \psi} \text{ eqMp} \quad \Big| = \text{EqMp} |\phi| |\psi| |D_1| |D_2| \\
\frac{\Gamma \vdash \phi}{\Gamma[\theta] \vdash \phi[\theta]} \text{ TypeSubst} \quad \Big| = (\lambda \alpha_1 : \text{type} \dots \lambda \alpha_m : \text{type}. |D|) |A_1| \dots |A_m| \\
\frac{\Gamma \vdash \phi}{\Gamma[\sigma] \vdash \phi[\sigma]} \text{ TermSubst} \quad \Big| = (\lambda x_1 : ||B|| \dots \lambda x_n : ||B_n||. |D|) |M_1| \dots |M_n|
\end{array}$$

### 3.3.2 Correctness

*Completeness* is to guarantee that the translated HOL terms have correct types. While *soundness* states that if a proof term is well-typed in Dedukti, the proofs must be correct in HOL Light. The *completeness* and *soundness* gives the correctness of a translation and was proved in [5].

### 3.3.3 Holide2

Recent updates of the OpenTheory format as shown in Table 3.3 requires an update of the Holide program (although users can still use the old version of Holide by limiting the output of OpenTheory to version 5). Holide uses a modular translation of higher order logic to Dedukti which makes the translation possible to extend [5]. The following shows an extension of Holide's translation. In addition, the following constants were added to Holide2 for to translate proofs using sym, trans and proveHyp.

- **Sym** :  $\Pi \alpha : \text{type}. \Pi x, y : \alpha. \text{proof}(\text{eq bool } x \ y) \rightarrow \text{proof}(\text{eq bool } y \ x)$
- **Trans** :  $\Pi \alpha : \text{type}. \Pi x, y, z : \text{term } \alpha. \text{proof}(\text{eq } \alpha \ x \ y) \rightarrow \text{proof}(\text{eq } \alpha \ y \ z) \rightarrow \text{proof}(\text{eq } \alpha \ x \ z)$
- **ProveHyp** :  $\Pi x, y : \text{term bool}. \text{proof } x \rightarrow \text{proof } y \rightarrow \text{proof } y$

$$\begin{array}{|l}
\frac{\Gamma \vdash \varphi = \psi}{\Gamma \vdash \psi = \varphi} \text{ sym} \quad \Big| = \text{Sym} |A| |t_1| |t_2| \\
\frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u} \text{ trans} \quad \Big| = \text{Trans} |A| |x| |y| |z| |D_1| |D_2|
\end{array}$$



$$\left| \frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma \cup (\Delta \setminus \{\varphi\}) \vdash \psi} \text{proveHyp} \right| = \text{ProveHyp} |x||y||D_1|\lambda x : ||t_1||.|D_2|$$

Similar as Holide, Holide2 takes HOL proofs in the OpenTheory article format (.art) of version 6 as input and outputs a Dedukti file (.dk). A comparison of Holide1 and Holide2 is shown in Table 6.2 and Table 6.3 in Chapter6

## Chapter 4

# Reverse Engineering of HOL Proofs

The relationship between classical inference systems and constructive (a.k.a intuitionistic) inference systems has been a longstanding field of research. The concept of intuitionism dates back to the discovery by Brouwer that classical mathematics do not correspond to the intuition we have of the mathematical objects. The discussion of classical and constructive reasoning continues after the discovery of some paradoxes and inconsistencies [14], especially Gödel's incompleteness theorem. Proving by contradiction in classical inference systems was also considered to be one of the sources of problems. In addition, classical reasoning would also lead to the lose of the witness property [12]. This showed the limitations of classical inference systems. Some thought classical reasoning should be allowed as long as it was finitistically justified while others insist to avoid non-constructive proofs [14]. In the middle the debate of intuitionism and classficism lies the law of excluded middle (LEM). Depending on whether there is a proof for  $P \vee \neg P$  for any  $P$  or not, inference systems are classified in classical logic (if there is such a proof) or constructive logic. A proof is regarded constructive when it is not using the law of excluded middle. Constructive proofs can also be considered classical proofs without using the law of excluded middle rule. Logical Framework provides a means to present a logic as a signature in such a way that provability of a formula in the original logic reduces to a type inhabitation problem in the framework type theory. In Logical Framework, proofs are defined using definitions of logical constants, connectives, quantifiers and axioms. Using a Logical Framework, the theories are made explicit, making it possible to investigate which part of the theory is needed for each given proof. This analysis is called "reverse engineering proofs". For instance, it is possible to investigate which proofs do not use the excluded middle. If so, these proofs can be expressed in the weaker constructive system. This internship project aimed at such automatic analysis for higher order logic by picking constructive-compatible definitions and transform proofs using those definitions.

An alternative view of this is to consider classical proofs as constructive proofs by embedding classical constants, connectives and quantifiers in constructive logic. Embedding methods of similar fashion are usually referred to as "double negation translation" or "negative translation". Double negation translation has a history back to 1925 from Kolmogorov's

translation [24]. Pioneering work also includes Gödel’s translation in 1933 and Gentzen’s translation in 1936 [17]. In 2013, another translation was provided by Dowek [12]. Most recently, Gilbert and Hermant provided a minimal translation for First Order Logic Sequent Calculus as morphism [16]. A morphism is to define an embedding of classical connectives, constants and quantifiers so that formulas that are provable in classical logic has its transformed formula provable in constructive logic.

## 4.1 Double Negation Translation

Negative translation has a history back to Kolmogorov’s translation [24], following that was Gödel’s translation in 1933 [17] and Gentzen’s translation in 1936. Most recently, Gilbert and Hermant provided a minimal translation for First Order Logic Sequent Calculus [16].

The first double negation translation was proposed by Kolmogorov by putting a double negation in front of every subformula [14]. Following Kolmogorov, a similar translation was presented by Gödel and Gentzen [14, 17, 22]<sup>1</sup>. Later in 1950s, Kuroda proposed another translation [25]. Most recently, Gilbert and Hermant presented a minimal translation [16]. The translation has been proved to be minimal for formulas that are not in the form of  $A_0 \wedge A_0 \wedge \dots \wedge A_n$ , namely barred formulas [16]. Barred formulas are those not in the form of  $A_0 \wedge A_0 \wedge \dots \wedge A_n$ . A translation which depends on the subformulas of the formula to be translated is proof-dependent. Although the translation varies, the development of double negation translation aims at simpler and more efficient translation without losing properties. Table 4.1 sums up some properties. The entry “Morphism” refers to whether a translation is a connective to connective translation making it a morphism<sup>2</sup>. However all of the translations requires universal quantifier and implication to be primitively defined. This led to the change of the kernel of OpenTheory HOL for the sake of double negation translation.

---

<sup>1</sup>Note that Gödel’s original translation also puts double negation in front of the translated terms of implication.

<sup>2</sup>Kuroda’s translation has an additional double negation in front of the whole proposition, making it not a morphism.

Property	Gödel-Gentzen	Kuroda	Gilbert-Hermant
Translation	$\perp^{gg} = \perp$ $\top^{gg} = \top$ $P^{gg} = \neg\neg P$ , for P atomic $A \wedge^{gg} B = A^{gg} \wedge B^{gg}$ $A \vee^{gg} B = \neg\neg(A^{gg} \vee B^{gg})$ $A \Rightarrow^{gg} B = (A^{gg} \Rightarrow B^{gg})$ $\neg^{gg} A = \neg A^{gg}$ $\forall^{gg} x A = \forall x A^{gg}$ $\exists^{gg} x A = \neg\neg\exists x A^{gg}$	$\perp^{ku} = \perp$ $\top^{ku} = \top$ $P^{ku} = P$ , for P atomic $A \wedge^{ku} B = A^{ku} \wedge B^{ku}$ $A \vee^{ku} B = (A^{ku} \vee B^{ku})$ $A \Rightarrow^{ku} B = (A^{ku} \Rightarrow B^{ku})$ $\neg^{ku} A = \neg A^{ku}$ $\forall^{ku} x A = \forall x \neg\neg A^{ku}$ $\exists^{ku} x A = \exists x A^{ku}$	$\perp^m = \perp$ $\top^m = \top$ $P^{ku} = P$ , for P atomic $A \wedge^m B = A^m \wedge B^m$ $A \vee^m B = \neg\neg(A^m \vee B^m)$ $A \Rightarrow^m B = A^m \Rightarrow \neg\neg B^m$ $\neg^m A = \neg A^m$ $\forall^m x A = \forall x \neg\neg A^m$ $\exists^m x A = \neg\neg\exists x A^m$
Proof-dependent	No	No	Yes
$\vdash^c  A  \rightarrow \vdash^i  A $	Yes	Yes	Yes
$\Gamma \vdash^c  A  \rightarrow \Gamma \vdash^i  A $	Yes	Yes	only if A is barred
Morphism	No	No	Yes
Minimal Translation	No	No	Yes

Table 4.1: Properties of Translation Compared

## 4.2 Kernel Hacking for Reverse Engineering of HOL Proofs

There are several reason for removing equality as primitive symbols. The first and the most important being that all theoretical results of double negation translation so far are based on universal and implication. Having universal quantifier and implication would also make the system fundamentally more similar as other proof assistants which would make it easier to transport proofs between each other. In addition, such change would bring the system of HOL Light and Dedukti closer for further optimisation of proof checking since Dedukti's logic foundation is based on universal quantifier and implication. Several attempts have been made to modify he kernel but none of them lead to a successful result by the deadline of the project. Among them HOLALA is to be presented in Chapter 5 and the rest are attached in Appendix C.

# Chapter 5

## HOLALA

As introduced in Chapter 2, HOL Light has a reliable kernel where equality and related primitive inference rules are defined. Based on the logic kernel, other connectives and constants were introduced as axioms followed by derived inference rules. HOLALA is a modified version of OpenTheory HOL Light where the kernel consists of three primitive logic symbols and their corresponding inference rules with some modification of OpenTheory HOL Light's proof logging methods.

### 5.1 Kernel Hacking

#### 5.1.1 Logic Kernel

$Q_0$	OpenTheory HOL Light	HOLALA
$=$	primitive	primitive
$\Rightarrow$	$\lambda pq.(p = (p \wedge q))$	primitive
$\forall$	$\lambda p.A = \lambda p.\top$	primitive
$\exists$	$\neg \forall x \neg A$	$\lambda p \forall q.(\forall x.px \Rightarrow q) \Rightarrow q$
$\top$	$= = =$	$\forall x.(x \Rightarrow x)$
$\perp$	$\lambda x.T = \lambda x.x$	$\forall p.p$
$\wedge$	$\lambda x \lambda y(\lambda g.g \top \top = \lambda g.gxy)$	$\lambda pq.(\lambda f.fpq) = \lambda f.f \top \top$
$\vee$	$\lambda xy.\neg(\neg x \wedge \neg y)$	$\lambda pq.\forall.(p \Rightarrow r) \Rightarrow ((q \Rightarrow r) \Rightarrow r)$
$\neg$	$\lambda x.\top = \lambda x.x$	$\lambda p.p \Rightarrow \perp$
$\exists$	$\neg \forall x \neg A$	$\lambda p.\forall q.(\forall x.px \Rightarrow q) \Rightarrow q$
$\Leftrightarrow$	$=$	$=$
$\exists!$	$(\exists p) \wedge (\forall xy.px \wedge py \Rightarrow x = y)$	$(\exists p) \wedge (\forall xy.px \wedge py \Rightarrow x = y)$

Table 5.1: Primitive and Axiomatic Definitions of Connectives and Constants Comparison

The Logic kernel of OpenTheory HOL Light has some modification from HOL Light. HOLALA can be considered a modified version of OpenTheory HOL Light. HOLALA does not only take equality as primitive symbol but also implication and universal quantification. The logic kernel consists of three primitive logic symbols defined based on implication and universal quantifier, together with their corresponding inference rules. Table 5.1 presents the

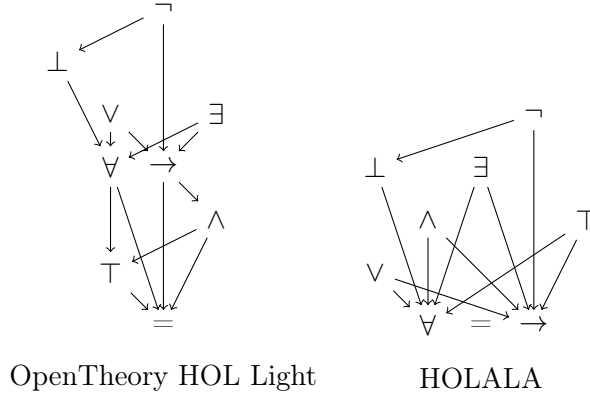


Figure 5.1: Dependency of Constants and Connectives Analysis

constants of the logic kernel of HOLALA in comparison with  $Q_0$  and OpenTheory HOL Light. Compared with OpenTheory HOL Light, HOLALA also changed the definition of  $\top$ , and  $\wedge$ , making as many definitions of logic symbol as possible dependent on universal quantification and implication instead.

To summarise, Table 5.1 presents a comparison with  $Q_0$ , OpenTheory HOL Light and HOLALA. Figure 6.5 shows the dependency analysis of OpenTheory HOL Light and HOLALA. Note that the graph omits equality as the definition symbol  $\equiv$  for the sake of simplicity. In practice, the equality plays the role of  $\equiv$  in every single axiom in HOLALA.

New primitive rules corresponding to implication and universal quantifier are therefore transported in the kernel as in Table 5.2. An immediate benefit of such change is that deduction rules directly related to  $\forall$  and  $\rightarrow$  got shortened. Deductive rules defined in *bool.ml* would therefore need to be reproofed. Table 5.3 shows the smallest proof<sup>1</sup> using the four basic rules which are outside of the kernel in OpenTheory but primitive (within the kernel) in HOLALA. Following that are some examples of derived rules as well as a comparison of the size of a proof of the Law of Excluded Middle. It is clear that smallest proofs using MP, DISCH, GEN and SPEC has reduced a dramatic amount of steps each. Deduced rules such as CONJ, IMP\_ANTISYM\_RULE and DISJ1 also reduced a significant amount of steps. However for rules not dependent on such changes, for example TRUTH<sup>2</sup>, the size stays the same. It is also noted that proving the law of excluded middle takes a significant number of steps instead of just one. More evaluation will be presented in the next section.

Although such change lead to reduction of size in most cases, users would lose the original definition of the  $\forall$  and  $\rightarrow$ . Proofs using these two definitions directly would therefore report errors. To fix it, these two definitions were proved as theorems after the introduction of the axiom of extensionality in HOLALA.

<sup>1</sup>Such proofs takes branches obtained by only one step of the axiom rule and directly followed by the inference rule.

<sup>2</sup>In both systems, TRUTH is taken as a proof using EQ\_MP, SYM, T\_DEF and REFL. None of these rules are directly related to the changes made by HOLALA.

$$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B} \text{MP}$$

$$\frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \forall x A} \text{GEN if } x \text{ is not free in } \Gamma$$

$$\frac{\Gamma \vdash B}{\Gamma \setminus \{A\} \vdash A \rightarrow B} \text{DISCH}$$

$$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} \text{SPEC}$$

Table 5.2: Additional Primitive Inference Rules of HOLALA

	OpenTheory HOL Light	HOLALA
MP	113	3
DISCH	106	2
GEN	23	2
SPEC	140	2
CONJ	57	33
DISJ1	158	23
IMP_ANTISYM_RULE	156	7
TAUT ( $\forall p.(p \vee \neg p)$ )	341	165

Table 5.3: Size of Deductive Rules Between OpenTheory HOL Light and HOLALA

### 5.1.2 Proof Logging

OpenTheory HOL Light modified the kernel of HOL Light and made it possible to export proofs by introducing a proof type. Instead of recording only the hypothesis and conclusion of proofs as in HOL Light, OpenTheory HOL Light records each step of HOL Light and deletes them after exporting to articles. HOLALA takes a similar approach but replaces the proof of the theorem Law of Excluded Middle (LEM) by a LEM\_proof as record of classicism. HOLALA therefore considers a proof being classical when it uses LEM. Instead of simply replacing the record of proofs by axiom proofs after exporting, HOLALA performs structural and statistical analysis and then takes the proof as a term of Lemma with corresponding results of analysis. Such analysis includes, detection of LEM, i.e. a sign of classical proof or constructive proof, size of proof, axioms used in a proof as well as classical and constructive lemmas used. Such improved proof logging and exporting technique leads to a search engine to be presented at the end of next chapter, namely ProofCloud.

## 5.2 Holide2x

HOLALA extends the logic kernel of HOL Light. Thus, a translation the following four inference rules is necessary. This is captured in another variant of Holide, namely Holide2x. The following is the modification of translation of types, terms and proofs. The translation

of HOL types remains the same but, to deal with universal quantifier and implication and their rules, Holide2x declares the `imp` and `forall` together with `MP`, `DISCH`, `GEN` and `SPEC` as constants.

### 5.2.1 HOL Terms

Similar to equality, universal quantification is also of polymorphic type. The translation of terms using universal quantification and existential quantification would therefore then be:  $|(M : A)| = \text{forall}|A||M|$  and  $|M \implies N| = \text{imp}|M||N|$ .

- `imp`:  $\text{term bool} \rightarrow \text{term bool} \rightarrow \text{term bool}$
- `forall`:  $\Pi \alpha : \text{type} \rightarrow \text{term}(\text{arr } \alpha \text{ bool}) \rightarrow \text{term bool}$
- $|\rightarrow| = \text{imp}$
- $|(!A)| = \text{forall}|A|$

### 5.2.2 HOL Proofs

There are only four new inference rules introduced. The translation of proofs using these rules are in the same style as [5].

$$\left| \frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B} \text{MP} \right| = \text{MP}|A||B||\mathcal{D}_1||\mathcal{D}_2|, \text{ where } \mathcal{D}_1 \text{ and } \mathcal{D}_2 \text{ are the proofs of } A \rightarrow B \text{ and } A \text{ respectively.}$$

$$\left| \frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \forall x A} \text{GEN, if } x \text{ is not free in } \Gamma \right| = \text{GEN}|A||c'||\mathcal{D}'|, \text{ where } c' = \lambda x : ||A||.|c|, \mathcal{D} \text{ is a proof of } A[c/x] \text{ and } \mathcal{D}' = \lambda x : ||A||.|\mathcal{D}|$$

$$\left| \frac{\Gamma \vdash B}{\Gamma \setminus \{A\} \vdash A \rightarrow B} \text{DISCH} \right| = \text{DISCH}|A||B||\mathcal{D}'||\mathcal{D}|, \text{ where } \mathcal{D}' \text{ is a proof of } A \text{ and } \mathcal{D} \text{ is a proof of } B$$

$$\left| \frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} \text{SPEC} \right| = \text{SPEC}|A||t'|u||\mathcal{D}|, \text{ where } t' = \lambda x : ||A||.|t| \text{ and } \mathcal{D} \text{ is a proof of } B \text{ respectively.}$$

The four constants `MP`, `DISCH`, `GEN` and `SPEC` introduced are as below.

`MP`:  $\Pi p : \text{term bool} . \Pi q : \text{term bool} . \text{proof}(\text{imp } p \ q) \rightarrow \text{proof } p \rightarrow \text{proof } q$

`DISCH`:  $\Pi p : \text{term bool} . \Pi q : \text{term bool} . \text{proof } p \rightarrow \text{proof } q \rightarrow \text{proof}(\text{imp } p \ q)$

`GEN`:  $\Pi \alpha : \text{type} . \Pi p' : (\text{term } \alpha \rightarrow \text{term bool}) . \Pi x : \text{term } \alpha . \text{proof}(p' \ x) \rightarrow \text{proof}(\text{forall } \lambda x . p' \ x)$

`SPEC`:  $\Pi \alpha : \text{type} . \Pi t : (\text{term } \alpha \rightarrow \text{term bool}) . \Pi u : \text{term } \alpha . \text{proof}(\text{forall } \alpha \ t) \rightarrow \text{proof}(t \ u)$

This extended version of Holide completes the workflow of proof checking of OpenTheory packages. proofs in HOLALA can be exported to new article files (.artx) and then translated as Dedukti files. Detailed evaluation and proof checking are to be presented in Chapter 6.



## Chapter 6

# Proof Analysis and Evaluation

In this section, the workflow of this project is first illustrated. Following that are some structural and statistical analysis of OpenTheory using Holide, HOLALA and Dedukti. In addition, specification of ProofCloud will be presented at the end of the chapter.

### 6.1 Workflow

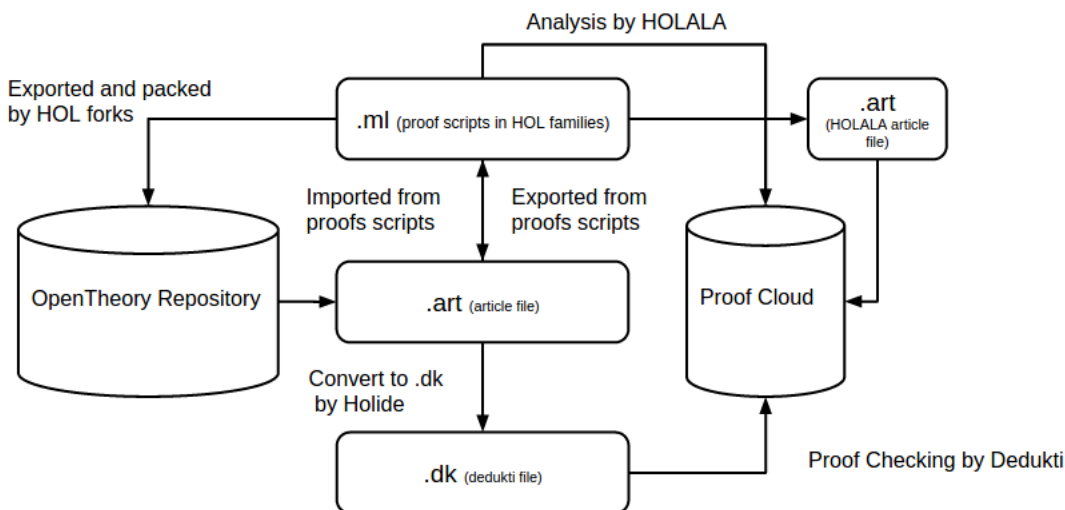


Figure 6.1: Workflow of HOLALA, Holide, OpenTheory and ProofCloud

This project uses many softwares of different versions. Before presenting the evaluation results, it is necessary to clarify the version correspondence and workflow of this project. On one hand, both OpenTheory repository and users can provide articles<sup>1</sup>. These article files are then converted to Dedukti files by Holide<sup>2</sup>. Then Dedukti performs proof checking and checking results goes into ProofCloud. On the other hand, proofs scripts can be loaded by HOLALA and perform proof analysis, then exported to HOLALA article files. These article files are then converted to Dedukti files by Holide<sup>2x</sup>. similarly, Dedukti conduct proof check-

<sup>1</sup>All articles in evaluation are assumed to be of OpenTheory version 6.

ing. Lastly, results from HOLALA and Dedukti get stored in ProofCloud. ProofCloud is not only a representation of proof analysis and proof checking results but also a search engine for a modified OpenTheory standard library. More details of ProofCloud will be presented at the end of this chapter.

Development of software makes correspondence of versions challenging. A significant amount of time of the project was to update Holide as a result of the changes of OpenTheory format and the changes of logic kernel of HOLALA. The recent updates of OpenTheory from version 5 to version 6 lead to the update of Holide from version 1 to version 2. Although users can still use the old version of Holide by limiting the output of OpenTheory to version 5<sup>2</sup>. Holide2 takes HOL proofs in the OpenTheory article format (.art) of version 6 as input and outputs a Dedukti file (.dk). After translation, dedukti takes the translated files as input and perform proof checking. HOLALA can be considered a modified version of OpenTheory HOL Light and generates articles of its own but can be considered an extended version 6 OpenTheory article. This brought about an alternative of Holide, namely Holide2x. All three of Holide1, Holide2 and Holide2x have been tested. The subsequent Table 6.1 displays the version correspondence clearly. Items in grey in the table are currently maintained by the author. The author has also contributed to the latest version of OpenTheory HOL Light on the import function making HOL Light import theorems from articles of version 6 correctly.

Article Generator / Repository	Article File Extension	Translation Tool and Version	Verification Tool
OpenTheory Repository 5	.art	Holide1	Dedukti
OpenTheory Repository 6	.art	Holide2	
HOLALA	.artx	Holide2x	

Table 6.1: Version Correspondance of OpenTheory, Holide and HOLALA

## 6.2 Proof Checking

The standard library in OpenTheory Repo grouped theorems into packages, including theorems of booleans, sets, lists, etc. The standard library (the base package, including 11 subpackages) was first verified in this project followed by all the packages in OpenTheory Repo. These tests were completed on a 64-bit Intel Core i5-4590 CPU @3.30GHz ×4 machine with 3.8GB RAM.

Above are two tables with the size of files and the time taken for translation and proof checking. Table 6.2 and Table 6.3 illustrates a comparison of version 1 and version 2 of

<sup>2</sup>Holide1 has checked the standard library of OpenTheory (version 5). The first release of Holide2 (in May 2015) could not translate articles in the lazylist package correctly due to a small mistake in translation of type variables. This has been corrected in most recent release of Holide2.

Package	Holide 1		Holide 2	
	Size (KB)	Time (s)	Size (KB)	Time (s)
base	1,436	19.35	1,194	19.42
cl	313	5.77	313	5.56
empty	0	0.20	0	0.00
gfp	136	1.42	112	1.35
lazy-list	1,390	31.43	1,391	31.78
modular	45	1.13	37	0.37
natural-bits	162	1.43	132	1.39
natural-divides	193	2.10	157	1.94
natural-fibonacci	130	1.31	108	1.24
natural-prime	140	1.46	116	1.34
parser	240	3.22	204	3.15
probability	26	0.30	23	0.23
stream	75	0.75	63	0.73
word10	86	0.76	71	0.62
word12	88	0.79	72	0.75
word16	131	1.60	107	0.77
word5	77	0.70	64	1.56
<b>Total</b>	<b>4,668</b>	<b>73.73</b>	<b>4,377</b>	<b>72.21</b>

Table 6.2: Size of Article Files and Translation Time

Package	Dedukti (Holide 1)		Dedukti (Holide 2)	
	Size (KB)	Time (s)	Size (KB)	Time (s)
base	4,681	10.63	4,440	9.74
cl	1,219	2.42	1,219	2.46
empty	0	0.00	0	0.00
gfp	400	0.73	375	0.65
lazy-list	5,718	13.31	5,717	13.11
modular	120	0.19	111	0.17
natural-bits	452	0.74	419	0.68
natural-divides	599	1.11	566	0.99
natural-fibonacci	378	0.67	354	0.60
natural-prime	408	0.72	388	0.65
parser	802	1.87	776	1.69
probability	72	0.12	69	0.11
stream	221	0.41	211	0.38
word10	234	0.38	216	0.29
word12	239	0.40	220	0.35
word16	396	0.80	364	0.36
word5	207	0.33	192	0.72
<b>Total</b>	<b>16,146</b>	<b>34.83</b>	<b>15,637</b>	<b>32.95</b>

Table 6.3: Size of Dedukti Files and Proof Checking Time

translation by Holide and proof checking by Dedukti. Both article files and Dedukti files are compressed by gzip to reduce the effect of syntax formatting and whitespace. Translation and proof checking can be completed within two minutes and are considered to be efficient. However, there is little difference in terms of efficiency. In addition, the increase of size after translation is consistent with previous work [5].

For the first time, all OpenTheory packages passed proof checking by Holide2 and Dedukti. This announced the completion of proof checking of all OpenTheory packages available. The success of checking of all these packages also provides evidence that OpenTheory being a reliable platform for higher order proofs as well as the correctness of updates of Holide1 and Holide2. In addition, this project also provides evidence for the correctness of HOL Light kernel and proofs from a different perspective compared with [27].

### 6.3 Proof Analysis

As described in Chapter 1, HOLALA changed the proof logging technique of OpenTheory HOL Light and made it possible to do structural and statistical analysis before exporting to article files. HOLALA considers proofs using Law of Excluded Middle as classical proof and constructive otherwise. HOLALA is a modified version of OpenTheory HOL Light and integrates the standard library of 1199 proofs. Among them, there are 531 constructive proofs and 668 classical proofs making 44.29% of them constructive proofs.

Taking axioms and lemmas as one step and all deduction steps as one step, we can calculate the size of proofs (up to lemma). The size of proofs varies from a few steps to millions of steps. As illustrated in Table 6.4, compared with OpenTheory HOL Light, there is an overall reduction of size around 40.87%. This indicates that introducing frequently used inference rules as primitive inference rules would sharply reduce the size of proofs.

Analysis of OpenTheory proofs (Figure 6.2) shows that two primitive deduction rules (subst and eqmp) combined makes up to over 45% of all the rules used making it essential to further investigate the subst and eqmp. One reason is that the kernel of OpenTheory HOL Light is very small making INST and PINST, EQ\_MP and MK\_COMB widely used in bool.ml and equal.ml<sup>3</sup>. This inspires further work on proof optimisation, especially on the reduction of the use of the two commands, subst and eqMp, as described in Chapter 6.

In contrast, Figure 6.3 indicates that the introduction of MP, DISCH, GEN and SPEC has not only reduced total size of article files but also cut the percentage of subst and eqmp to around 24% resulting more balanced and less dependent deduction system. However, HOLALA's changes have little impact on appthm, trans and sym.

During the project the author also noticed that many common deducted inference rules are

---

<sup>3</sup>OpenTheory HOL Light takes  $\beta$ -conversion rule as primitive inferences rules which helps reducing the use of EQ\_MP and INST.

	OpenTheory HOL Light Count	HOLALA Count
subst	93330	27217
eqmp	92264	31234
appthm	52994	49712
proveHyp	47597	12782
betaConv	21370	8152
absThm	15024	9088
trans	26616	25337
refl	26644	25061
deductAntisym	9528	1246
sym	9401	8469
assume	16950	8376
mp	0	9644
disch	0	9455
gen	0	6344
spec	0	11327
Total	411718	243444

Table 6.4: Size and Frequency Analysis of Main Inference Rules of HOL Light and HOLALA Proofs

dependent on INST and INST\_TYPE (mostly used via PINST). Having INST or INST\_TYPE increased the complexity for proof searching automation. Such inference rule should be thought twice whether to be kept as primitive inference rule.

Since article files are compressed. A more practical way to compare proof size is to consider the size of the article files. Article files are where proofs are stored. The size of both article files and Dedukti files scaled down a lot. Article files of HOLALA are only around 23.63% on average the size of article files of OpenTheory<sup>4</sup>. Leading to an improvement of 41.81% in translation time. Size of Dedukti files got reduced to about 64.33% with a speed improvement of 38.04%.

## 6.4 ProofCloud

ProofCloud<sup>5</sup> is a proof retrieval engine for easy searching of verified higher order logic proofs<sup>6</sup>. It consists of a presentation of proofs and proof packages, the results of some statistical and structural analyses, together with their proof checking results. So far it has been populated with 1687 proofs from 6 packages of OpenTheory. Information of each proof and package is presented on separate webpages. Taking classical proofs as those using the axiom of choice,

<sup>4</sup>For fair comparison, article files and dedukti files are compressed with gzip separately and then the whole folder was compressed by as a tar.gz file. All article files are obtained from "/opentheory/article" directory. To remove the factor of optimisation by opentheory, articles are not combined to packages.

<sup>5</sup><http://airobert.github.io/proofcloud/>

<sup>6</sup>In this revisited version of internship report, ProofCloud is only about proofs and proof analysis results of OpenTheory. It has nothing to do with Holide2x and HOLALA anymore.

Main Inference Rules of OpenTheory Articles

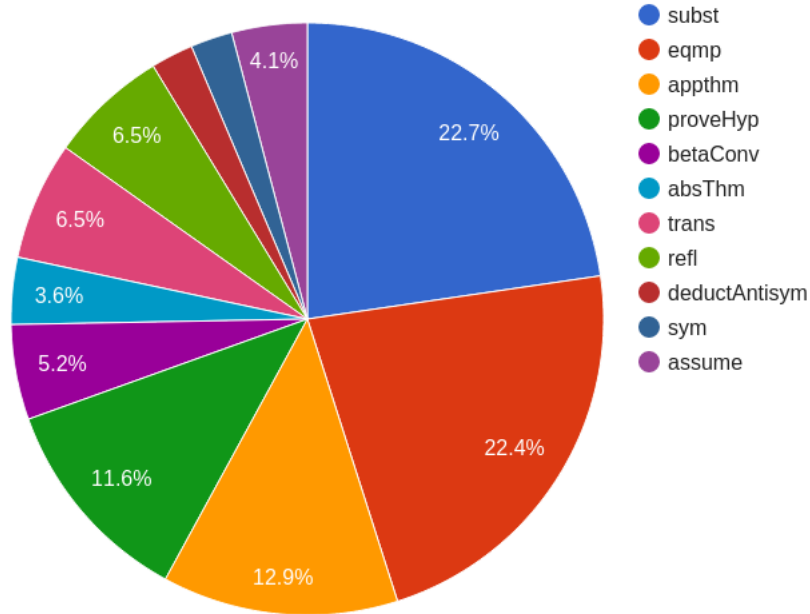


Figure 6.2: Frequency of Main Inference Rules of OpenTheory Articles

Main Inference Rules of HOLALA Articles

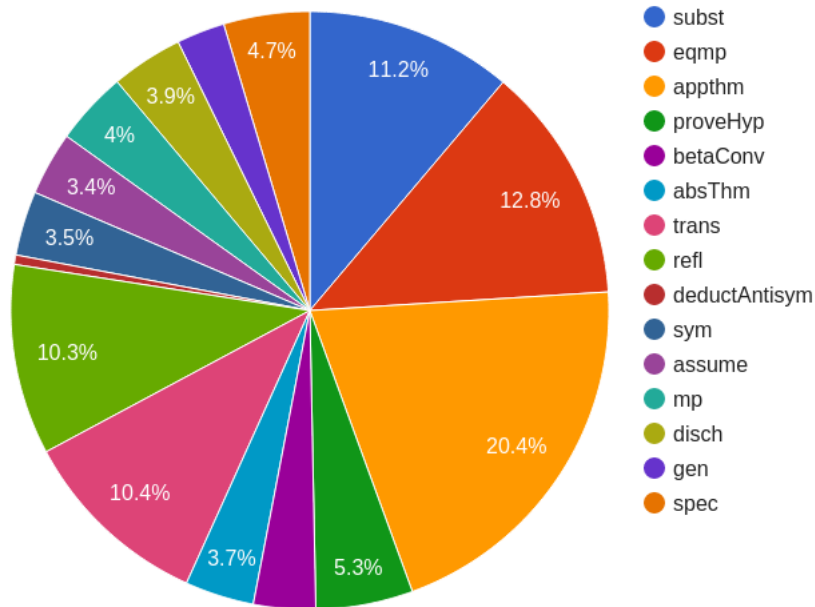


Figure 6.3: Frequency of Main Inference Rules of HOLALA Articles

	Size of Proof Files (KB)	Translation Time (s)
HOL Light	5,376	55.98
HOLALA	3,460	32.57
Comparison	Reduced to 64.36%	Improved by 41.81%

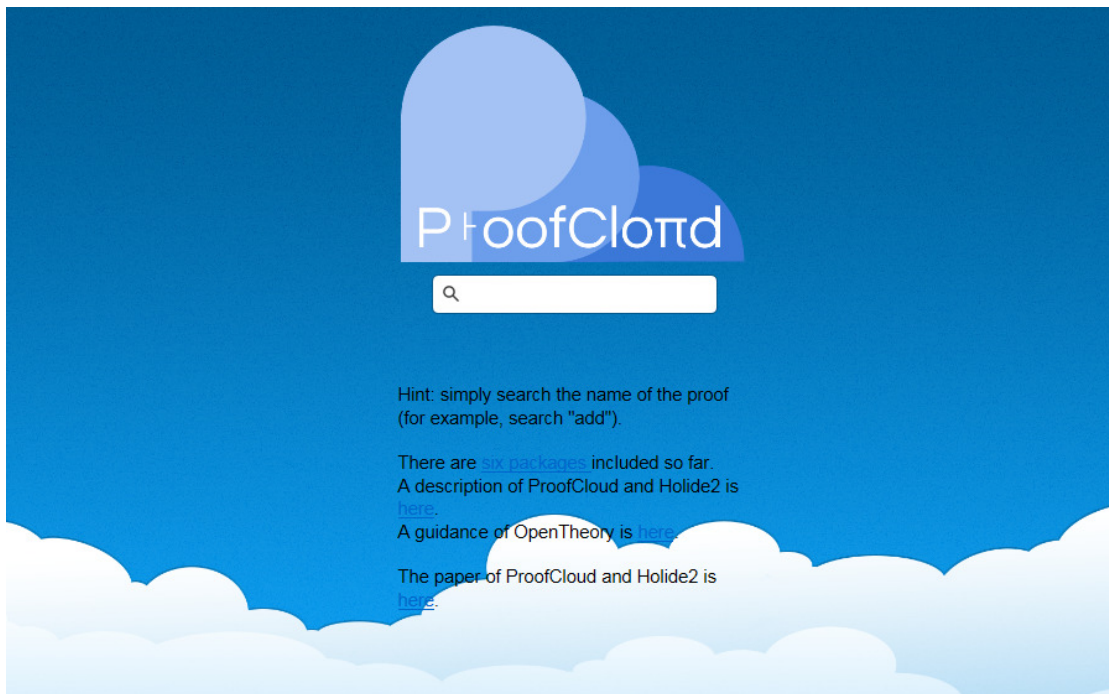
  

	Size of Dedukti Files (KB)	Proof Checking Time (s)
HOL Light	16,092	30.75
HOLALA	10,448	19.05
Comparison	Reduced to 64.92%	Improved by 38.04%

Table 6.5: Comparison of Translation and Proof Checking

ProofCloud can distinguish between classical proofs and constructive proofs. Furthermore, it tracks the origin of classicism, in other words, which classical lemmas<sup>7</sup> were used within the proof. With this ability, the amount as well as the percentage of classical proofs of a certain package is calculated and displayed on its (package) page. Users will also find a statistical analysis on the size of proofs and links to proof checking results. As far as the author knows, this is the only online proof retrieval engine of its kind.

Figure 6.4: The Interface of ProofCloud



So far ProofCloud includes six OpenTheory packages in total. These packages are *base* (the standard library with some subpackages), *stream*, *probability*, *natural-bits*, *natural-divides*

<sup>7</sup>To avoid confusion, all theorems used to prove the conclusion are referred to as lemmas when referring to a specific theorem.

and *natural-prime*. Other packages will be added in the near future. An analysis of the dependency of packages of OpenTheory is as shown in Figure 6.5. To load a package, it is required to load all the packages it depends on primarily. In addition, the package *modular* is a parametric theory and requires the *modular-witness* theory which defines a suitable signature. Similarly *gfp* is another parametric theory. These packages are being updated and therefore not included for now. The package *modular* and *gfp* and packages depending on them will be added to ProofCloud in the near future.

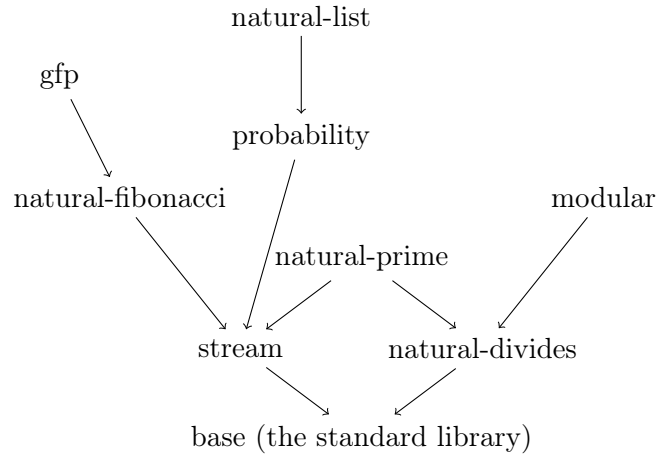



Figure 6.5: Dependency of Packages of OpenTheory

ProofCloud includes also the proof checking results for all packages of OpenTheory. The standard library of OpenTheory grouped theorems into packages, including theorems of booleans, sets, lists, etc. The standard library (the base package, including 11 subpackages) was first verified in this project, followed by all the packages in the OpenTheory repository. For the first time, Holide and Dedukti successfully translated and checked all packages in OpenTheory. Table 6.2 illustrates the size of OpenTheory proof article files and the time taken for translation. Table 6.3 represents the size of translated files and the time taken for proof checking by Dedukti. We name the translators as Holide 1 and Holide 2 respectively corresponding to OpenTheory version 5 and 6 respectively in Table 6.2 and Table 6.3. Both article files and Dedukti files are compressed by *gzip* to reduce the effect of syntax formatting and whitespace. These benchmarks were generated on a 64-bit Intel Core i5-4590 CPU @3.30GHz ×4 PC with 3.8GB RAM. However, there is little difference in terms of the size of article files and the efficiency between OpenTheory 5 and 6 for proof checking. The size of proof articles were reduced by around 7% while the proof checking time was reduced by around 5%. Figure 6.7 gives an example of the proof checking page of the stream package. In addition, Appendix D presents the specification of ProofCloud for further contribution.




Figure 6.6: A Proof Page of ProofCloud



Entry	Value
Name	ALL_F
Conclusion	!! ALL (x. F) I <=> NULL I
Constructive Proof	No
Axiom	<pre>!t. t \/\ ~t (\a. a = (\b. (\c. c) = (\c. c))) (\d. (\e. d e) = d)</pre>
Classical Lemmas	<ul style="list-style-type: none"> <li>• !x s. ~(x INSERT s = {})</li> <li>• !t. ~-t &lt;=&gt; t</li> <li>• !t. (t &lt;=&gt; T) V (t &lt;=&gt; F)</li> <li>• !p. (?x. ~p x) &lt;=&gt; ~(!x. p x)</li> <li>• !p. ~(!x. p x) &lt;=&gt; (?x. ~p x)</li> <li>• !p. ~(?x. p x) &lt;=&gt; (!x. ~p x)</li> <li>• !l. set_of_list l = {} &lt;=&gt; NULL I</li> <li>• !s t. ~(s = t) &lt;=&gt; (?x. x IN t &lt;=&gt; ~(x IN s))</li> <li>• !s. (?x. x IN s) &lt;=&gt; ~(s = {})</li> </ul>
Constructive Lemmas	<ul style="list-style-type: none"> <li>• T</li> <li>• !x y s. x IN y INSERT s &lt;=&gt; x = y V x IN s</li> <li>• !x y. x = y &lt;=&gt; y = x</li> <li>• !x y. x = y ==&gt; y = x</li> <li>• !h t. ~NULL (CONS h t)</li> <li>• !h t. set_of_list (CONS h t) = h INSERT set_of_list t</li> <li>• !x s. ~(x INSERT s = {})</li> <li>• !x s. x INSERT s = {y   y = x V y IN s}</li> <li>• !x. ~(x IN {})</li> </ul>

Figure 6.7: A Proof Checking Page of ProofCloud



Entry	Value
Proof Checking Engineer	Robert White
Time for verification	0.38(s)
Time for translation (if any)	0.73(s)
Software	Holide, Dedukti
Specification	64-bit with Intel @ Core i5-4590 CPU @ 3.30GHz * 4 and 3.8G memory on Ubuntu 15.04 running Ocaml 4.01 (and Camlp5 6.12)

[Back to package page](#)

## Chapter 7

# Conclusion and Future Work

### 7.1 Conclusion and Contribution

In this report, Chapter 2 and Chapter 3 are an exposition of the tools used during this internship: HOL-Light, Dedukti and Holide. To perform double negation translation, it is required to replace the primitive symbols of OpenTheory HOL Light kernel by implication and universal quantifier. Chapter 4 and Chapter 5 present a modification of the OpenTheory HOL-Light system, namely HOLALA, which meant to change the definition of the connectives and quantifiers, followed by the corresponding modifications of Holide. Chapter 6 presents proof analysis and the evaluation: statistical analysis shows that, 44% of the proofs are constructive. Structural analysis shows that introducing more symbols to the kernel leads to the reduction of proof size to around 64%, which result in an improvement of 41% in translation time and 38% in proof checking time. In addition, ProofCloud was developed as a representation of the analysis of HOL proofs. It is the only proof search engine of its kind.

### 7.2 Optimisation of proofs

Table 6.4 suggested that many common derived inference rules are dependent on INST and INST\_TYPE. Having INST or INST\_TYPE might increased the complexity for proof searching automation. It would also be necessary to analyze inference Rules and their dependency of all deduction rules for further optimisation. Another inference rule which worth examining is the BETA\_CONV rule. This rule built based on INST and BETA in (OpenTheory) HOL Light. While in OpenTheory, BETA\_CONV is taken as primitive inference rule. Having BETA\_CONV as primitive rule would not only reduce the size of proof but also reduce complexity for proof search. As a direct contribution, it can be expected that introducing techniques other than the combination of BETA, INST and EQ\_MP to deal with beta reduction would not only reduce the size of proof tree but also make inference clearer and easier. A solution for this is to introduce Deduction Modulo. Having Deduction Modulo would also allow introducing of constants and connectives independent of systems making defining equality based on universal quantifier and implication possible. This lead to the design of

HOL-Modulo.

### 7.3 HOL-Modulo

Although HOLALA is a reasonable attempt at removing equality as primitive rule, it lead to a more interesting question: why not transform HOL Light to a theorem prover taking advantage of Deduction Modulo. In theory, this would be one solution of replacing the equality in HOL Light. Instead of having beta reduction, defining equality up to  $\beta$  (instead of up to  $\alpha$ ) would also make proof shorter and more portable to Dedukti. There are also a small amount of code sharable between HOL-Modulo and Dedukti, making the integration of HOL-Modulo and Dedukti possible. Further than the analysis in Section 6.2, having equality up to  $\beta$  would replace the use of BETA\_CONV and reduce a significant amount of INST and INST\_TYPE and may provide a solution for removing INST and INST\_TYPE in proofs, which would benefit proof searching. A first step is to re-implement term comparison function. Following that it is required to introduce rewrite rules as well some rewriting algorithms for term conversion and proof construction. In addition, proof searching methods could be optimized. Further than that, it would be an possible to integrate HOL-Modulo with Dedukti making Dedukti not only a proof checker but also a theorem prover. Since Simple Type Theory can be expressed in Deduction Modulo [10], it is promising that HOL-Modulo be a step towards replacing the current kernels of HOL families and make ProofCloud a representation of rewriting rules for theorem proving.

### 7.4 HOL-Tableaux

Proof automation in interactive theorem proving is important since it reduces interaction and human labour and increase efficiency and usability of theorem provers. However, automated theorem proving methods in HOL Light such as ITAUT, TAUT, MESON\_TAC, ASM\_MESON\_TAC do not always produce effecient proofs. And sometimes proofs can be really huge, for example TAUT in Table 5.3. In some cases, proof searching can go really deep and time-consuming. Having noticed also that Sequent Calculus is equivalent to Natural Deduction [11] (See Table A.2) and Sequent Calculus has a lot similarity compared with Tableaux Methods, this project aims at taking advantage of Tableaux methods for better proof search and convert current proofs from HOL Light to better and simpler proofs. Some data analysis in this project gave a good hint that some proofs are too large and have to be simplified. Thus HOL-Tableaux would give these proofs a chance to get shorter in a smart way.

## 7.5 ProofCloud

The ProofCloud described in this revisited internship report differs from that of the original report. Originally ProofCloud was developed as a platform to represent some proof analysis results which were too big to be included as tables in the report. ProofCloud is now a complete and stand-alone platform that has nothing to do with HOLALA. Future work includes adding the remaining packages of OpenTheory to ProofCloud. Most recent updates of OpenTheory include also the names of proofs in packages. This would benefit the usability of ProofCloud. ProofCloud (version 3) is being developed with a new ontological representation using Web Ontology Language (OWL) for better representation of proofs and axioms and more.

# Appendix A

## Comparison of Sequent Calculus, Natural Deduction and HOL Light

Sequent Calculus, Natural Deduction are the two most widely used style of inference systems. Intuitionistic Natural Deduction can be derived in HOL Light. The following is an example proof in Sequent Calculus, Natural Deduction and HOL Light.

Sequent Calculus	Natural Deduction and HOL Light
$\frac{}{p \vdash p} \text{ axiom}$ $\frac{\frac{}{q \vdash q} \text{ axiom}}{p, q \vdash q} \text{ weakL} \rightarrow L$ $\frac{p, p \rightarrow q \vdash q \quad \wedge L}{(p \wedge (p \rightarrow q)) \vdash q} \rightarrow R$ $\frac{}{\vdash (p \wedge (p \rightarrow q)) \rightarrow q} \rightarrow q$	$\frac{}{p \wedge (p \rightarrow q) \vdash p \wedge (p \rightarrow q)} \text{ axiom}$ $\frac{p \wedge (p \rightarrow q) \vdash p \rightarrow q}{p \wedge (p \rightarrow q) \vdash p} \wedge \text{elim}_1 \rightarrow \text{elim}$ $\frac{p \wedge (p \rightarrow q) \vdash q}{\vdash (p \wedge (p \rightarrow q)) \rightarrow q} \rightarrow \text{intro}$ $\frac{}{p \wedge (p \rightarrow q) \vdash p \wedge (p \rightarrow q)} \text{ axiom}$ $\frac{p \wedge (p \rightarrow q) \vdash p \rightarrow q}{p \wedge (p \rightarrow q) \vdash p} \wedge \text{elim}_2 \rightarrow \text{elim}$ $\frac{p \wedge (p \rightarrow q) \vdash q}{\vdash (p \wedge (p \rightarrow q)) \rightarrow q} \rightarrow \text{intro}$

Table A.1: Example Proof of  $(p \wedge (p \rightarrow q)) \rightarrow q$  in Sequent Calculus, Natural Deduction and HOL Light

Table A.2 is a comparison of Sequent Calculus, Natural Deduction and HOL Light's logic. There is only one HOL Light primitive inference rule in the table (*ASSUME*). All the others are deduced rules (deduced from primitive rules) as marked in double bar. Note that the *Axiom* rule in Natural Deduction corresponds to the *ASSUME* rule in HOL Light. As for DISCH, A is not required to be in  $\Gamma$ . In addition, HOL Light does not allow repeated terms in its  $\Gamma$ . Thus no contraction rule for HOL Light.

	Sequent Calculus (LJ)	Natural Deduction (NJ)	HOL Light
Axiom	$\overline{\Gamma, A \vdash A}$ <i>axiom</i>	$\overline{\Gamma, A \vdash A}$ <i>axiom</i>	$\overline{\{A\} \vdash A}$ <i>ASSUME</i>
$\top$	$\overline{\Gamma \vdash \top}$ $\top_R$	$\overline{\Gamma \vdash \top}$ $\top_{intro}$	$\overline{\Gamma \vdash \top}$ <i>TRUTH</i>
$\perp$ Weak	$\frac{\Gamma \vdash}{\Gamma \vdash A}$ <i>weak<sub>R</sub></i>	$\frac{\Gamma \vdash \perp}{\Gamma \vdash A}$ $\perp_{elim}$	$\frac{\Gamma \vdash \perp}{\Gamma \vdash A}$ <i>CONTR</i>
	$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}$ <i>weak<sub>L</sub></i>		$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$ <i>ADD_ASSUM</i>
	$\overline{\Gamma, \perp \vdash \Delta}$ $\perp_L$		
$\wedge$		$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$ $\wedge_{elim1}$ $\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$ $\wedge_{elim2}$	$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$ <i>CONJUNCT1</i> $\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$ <i>CONJUNCT2</i>
	$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$ $\wedge_L$		
	$\frac{\Gamma \vdash A}{\Gamma \vdash A \wedge B}$ $\wedge_R$	$\frac{\Gamma \vdash A}{\Gamma \vdash A \wedge B}$ $\wedge_{intro}$	$\frac{\Gamma \vdash A}{\Gamma \cup \Delta \vdash A \wedge B}$ <i>CONJ</i>
$\vee$	$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$ $\vee_L$		

	$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \text{velim}$	$\frac{\Gamma \vdash A \vee B \quad \Gamma_1, A \vdash C \quad \Gamma_2, B \vdash C}{\Gamma \cup (\Gamma_1 \setminus \{A\}) \cup (\Gamma_2 \setminus \{B\}) \vdash C} \text{DISJ\_CASES}$
	$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{R1}$	$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \text{DISJ1}$
	$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{R2}$	$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \text{DISJ2}$
	$\frac{\Gamma \vdash A \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta} \rightarrow_L$	
$\rightarrow$	$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow \text{elim}$	$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B} \text{MP}$
	$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_R$	$\frac{\Gamma \vdash B}{\Gamma \setminus \{A\} \vdash A \rightarrow B} \text{DISCH}$
	$\frac{\Gamma \vdash A}{\Gamma, \neg A \vdash} \neg_L$	$\frac{\Gamma \vdash \neg A}{\Gamma \vdash \neg A} \text{NOT\_ELIM}$
$\neg$	$\frac{\Gamma, \neg A \vdash}{\Gamma, A \vdash} \neg \text{elim}$	$\frac{\Gamma \vdash A \rightarrow \perp}{\Gamma \vdash \neg A \equiv \perp} \text{EQF\_ELIM}$
	$\frac{\Gamma, A \vdash}{\Gamma \vdash \neg A} \neg_R$	$\frac{\Gamma \vdash A \rightarrow \perp}{\Gamma \vdash \neg A} \text{NOT\_INTRO}$
$\forall$	$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta} \forall_L$	$\frac{\Gamma \vdash A \equiv \perp}{\Gamma \vdash A \equiv \perp} \text{EQF\_INTRO}$
	$\frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \forall x A} \forall_R$	$\frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \forall x A} \text{GEN if } x \text{ is not free in } \Gamma$
	$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} \forall \text{elim}$	$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} \text{SPEC}$

$\exists$	$\frac{\Gamma, A[c/x] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta} \exists_L$		
	$\frac{\Gamma \vdash A[c/x]}{\Gamma \vdash \exists x A} \exists_R$	$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A} \exists_{intro}$	$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A} \exists_{exists}$
		$\frac{\Gamma \vdash \exists x A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \exists_{elim}$ if x is not free in $\Gamma$ and B	$\frac{\Gamma \vdash \exists x A[x] \quad \Delta, A \vdash B}{\Gamma \cup \Delta \setminus \{A[v/x]\} \vdash B} \text{CHOOSE}$ if v is not free in $\Delta \setminus \{A[v/x]\}$ , A or B)
	$\frac{\Gamma \vdash A \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta} \text{cut}$		
Contr	$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} \text{contr}_L$		

Table A.2: Inference System Comparison



# Appendix B

## Proof of Law of Excluded Middle

### B.1 Law of Excluded Middle

Generally speaking, classical Logic has the law of excluded middle (LEM) as an axiom while the conclusion of LEM is not provable in constructive logic. LEM states that a proposition is either true or false. In other words, the statement  $P \vee \neg P$  is a tautology:

**Axiom 10** (Law of Excluded Middle (LEM)).  $\forall p.p \vee \neg p$

In OpenTheory HOL Light, LEM is derived from functional extensionality and the axiom of choice.

### B.2 Diaconescu's Proof of Law of Excluded Middle

It has been proved in [8] that the axiom of choice implies the law of excluded middle using separation and extensionality. The choice symbol is as introduced in Section 2.2.3.

**Axiom 11** (Axiom of Choice).  $\forall P \forall x : A.Px \Rightarrow P((@)P)$

**Axiom 12** (Axiom of Extensionality).  $\forall f : (A \rightarrow B) \forall g : (A \rightarrow B).(\forall x.f x = g x) \Rightarrow f = g$

The proof idea can be summarized as follows. Taking  $A = \{n \in \mathbb{N} : n = 0 \vee (n = 1 \wedge \phi)\}$  and  $B = \{n \in \mathbb{N} : n = 1 \vee (n = 0 \wedge \phi)\}$ . When  $\phi$  is true,  $A = \{0, 1\}$  and  $B = \{0, 1\}$  but when  $\phi$  is false,  $A = \{0\}$  and  $B = \{1\}$ . Thus, in either case, neither of  $A$  nor  $B$  would be empty. Suppose  $f$  is a choice function so that  $f(A) \in A$  and  $f(B) \in B$ . There are two cases,  $A = B$  or  $A \neq B$ . On one hand, if  $A = \neg B$ , then  $\phi$ . On the other hand, when  $A \neq B$ , suppose  $\phi$ , using extensionality, we can get  $f A = f B$ . However this would give us contradiction with the hypothesis. Therefore,  $\neg\phi$ . Since the two cases infers  $\phi$  and  $\neg\phi$  respectively, we can conclude that  $\phi \vee \neg\phi$ . This proof was implemented by Mark Adams in HOL Zero and there is an adapted version in HOL Light. A shorter proof is to be released in the most recent version of HOL Zero<sup>1</sup>.

---

<sup>1</sup>This shorter proof was in an email with Mark on 31st May 2015.



## B.4 Proof of $\neg\neg$ LEM

Most double negation translation translate LEM to  $\neg\neg$ LEM or similar form, which is provable in constructive logic as presented follows:

$$\begin{array}{c}
 \frac{\frac{\frac{\neg(p \vee \neg p) \vdash \neg(p \vee \neg p)}{\neg(p \vee \neg p), \neg p \vdash \neg(p \vee \neg p)}}{\neg(p \vee \neg p), \neg p \vdash (p \vee \neg p) \rightarrow \perp}}{\neg(p \vee \neg p), \neg p \vdash \perp} \quad \frac{\frac{\frac{\neg p \vdash \neg p}{\neg(p \vee \neg p), \neg p \vdash \neg p}}{\neg(p \vee \neg p), \neg p \vdash p \vee \neg p}}{\neg(p \vee \neg p), \neg p \vdash \perp} \quad \frac{\frac{\frac{p \vdash p}{\neg(p \vee \neg p), p \vdash p}}{\neg(p \vee \neg p), p \vdash p \vee \neg p}}{\neg(p \vee \neg p), p \vdash p \vee \neg p} \quad \frac{\frac{\frac{\neg(p \vee \neg p) \vdash \neg(p \vee \neg p)}{\neg(p \vee \neg p), p \vdash \neg(p \vee \neg p)}}{\neg(p \vee \neg p), p \vdash (p \vee \neg p) \rightarrow \perp}}{\neg(p \vee \neg p), p \vdash \perp} \\
 \frac{\frac{\frac{\neg(p \vee \neg p), \neg p \vdash \perp}{\neg(p \vee \neg p) \vdash \neg p \rightarrow \perp}}{\neg(p \vee \neg p) \vdash \perp} \quad \frac{\frac{\frac{\neg(p \vee \neg p), p \vdash \perp}{\neg(p \vee \neg p) \vdash \neg p}}{\neg(p \vee \neg p) \vdash \perp}}{\frac{\frac{\neg(p \vee \neg p) \vdash \perp}{\vdash \neg\neg(p \vee \neg p)}}{\vdash \neg\neg(p \vee \neg p)}}
 \end{array}$$

Having  $\neg\neg$ LEM proved, we are able to replace the LEM in the original proofs.

## Appendix C

# Attempts of reverse engineering of HOL Light Proofs

In this chapter, several attempts are presented to change the kernel. There are several reasons for removing equality as primitive symbols. The first and the most important reason being all theoretical results of double negation translation so far are for first order logic and are based on universal and implication. Having universal quantifier and implication would also make the system fundamentally more similar as other proof assistants which would make it easier to transport proofs between each other. In addition, such change would bring the system of HOL Light and Dedukti closer for further optimisation of proof checking since Dedukti's logic foundation is based on universal quantifier and implication. We will present the design of HOL-intermediate, HOLIU and HOLbot. Among them, HOL-intermediate is a premier attempt of making as many symbols depending on implication and universal quantifier as possible. HOLIU has universal quantifier and implication as primitive connectives as well as the equivalent symbol ( $\equiv$ ) but this experiment was not proved to be successful. HOLbot is another view of the logic kernel with all common logic symbols taken as primitive. Both HOLIU and HOLbot inspired the proposal of HOL-Modulo. Figure C.1 illustrates the dependency of constants and connectives of OpenTheory HOL Light, HOL-intermediate and HOLIU. In contrast, not aiming at removing equality, HOLbot takes all constants and connectives as primitive symbols for reduction of proof size and proof automation.

Although none of them is able to remove equality from the kernel and introduce universal and implication as the only primitive quantifiers, these attempts are good exercises for kernel hacking of (OpenTheory) HOL Light. Also these attempts result in deeper understanding of the kernel and implementation details as well as the analysis of dependency of constants, connectives and inference rules and inspire further development of this project.

OpenTheory HOL Light has the same logic foundation (a.k.a logic kernel) as HOL Light but with a proof logging method in its kernel. It takes equality as the only primitive symbol and defines all the other logic connectives and constants based on it. The kernel includes also some primitive inference rules (as in Table 2.3) where all the other inference rules (as in Table 2.4) are based on.

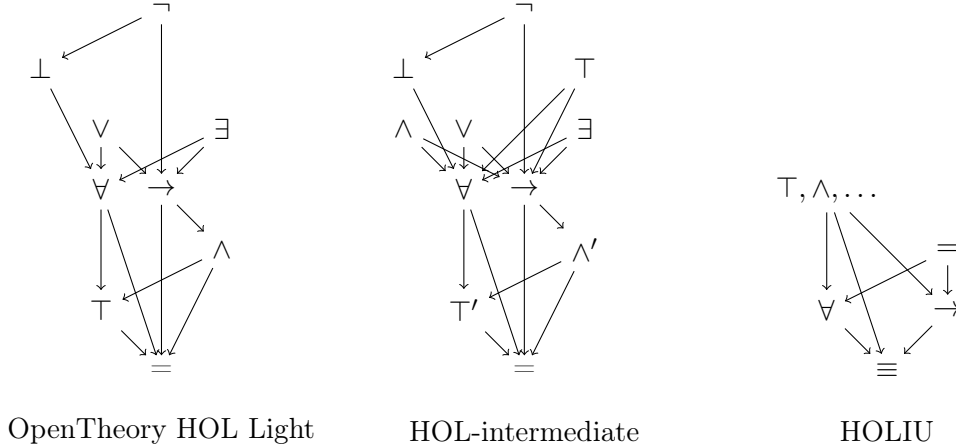


Figure C.1: Constants and Connectives Dependency Analysis and Design

## C.1 HOL-intermediate

HOL-intermediate was the premier experiment in this project. As a very first attempt to change the kernel, HOL-intermediate kept equality as primitive rule but tries to build all the rest on top of the universal quantifier and implication. Instead of having the two connectives,  $\top$  and  $\wedge$ , depending on equality. These symbols were introduced on top of  $\rightarrow$  and  $\forall$  after changing  $\top$  and  $\wedge$  to  $\top'$  and  $\wedge'$ . This way, there are as symbols as possible depending on  $\rightarrow$  and  $\forall$ . All the derived inference rules corresponding to  $\top$  and  $\wedge$  were then proved again. The implementation of HOL-intermediate requires knowledge of the kernel of HOL Light, especially the understanding of introducing new symbols, proving deduction rules as well as dependency analysis. The successful load of HOL libraries marked the completion of this attempt. With the experience of HOL-intermediate, we then tried to introducing universal quantifier and implication, which lead the design and implementation of HOLALA as presented in Chapter 5.

## C.2 HOLIU

HOLIUI is another attempt to have universal quantifier and implication as primitive symbol. However, after realising that it is hard to remove equality from the kernel, in this attempt,  $\equiv$  was introduced instead of  $=$ , making it possible to define new symbols without using  $=$  as primitive symbol. This way,  $\forall$  and  $\rightarrow$  depend only on *equiv* and were successfully introduced as primitive symbol. Following that, we define equality as the Leibniz's equality:

**Definition 1.**  $= \equiv \lambda xy. \forall P. Px \rightarrow Py$

Correspondingly, this leads to the introduction of the EQUIV rule:

$$\frac{\Gamma \vdash A \equiv B}{\Gamma \vdash A = B} \text{EQUIV}$$

As a result, we lose almost all primitive inference rules including the  $\eta$  reduction, which is key to  $\beta$ -conversion. This also results in the failure of the introduction of the rest of all inference rules. Thus, this approach was proved unsuccessful. Although it would not be

working, if terms are equivalent to *beta* and we take rewrite rules instead of  $\equiv$  for the use of definition, there might be a way to remove equality from the kernel. This attempt leads to the design of HOL-Modulo.

### C.3 HOLbot

It was noticed in Table 5.3 and Table 6.2 that proofs in HOL-Light can be really big in size. One of the reasons is that the size of the kernel is really small, resulting in many derived inference rules. During this project, it was noticed that a small kernel benefits proof checking and verification but can make proofs big. Table 5.3 shows that introducing universal quantifier and implication can reduce the size of derived inference rules. Having more primitive rules would enlarge the kernel but shorter proofs can be expected. HOLbot follows from this idea. It has been also noticed during discussions that Sequent Calculus has some similarities as Tableaux Method. Since Tableaux Method provides a fully automated proof searching approach, it would be interesting to know if extending the kernel would reduce the size of proofs discovered automatically. HOLbot will be further explored as HOL-Tableaux.

# Appendix D

## The Specification of ProofCloud

The following attributes are included in the webpages of each package:

### D.1 The Specification of ProofCloud

The following attributes are included in the webpages of each package:

- Package name
- Author of package
- Subpackages
- Date retrieved
- Total number of proofs
- Number of constructive proofs
- Number of classical proofs
- Percentage of constructive proofs
- Size of constructive proofs on average
- Size of classical proofs on average
- List of proofs (names and conclusions)
- Comments

For each package, there is also a page for verification (proof checking) information with the following entries:

- Software engineer for verification
- Software for verification
- Translation time
- Verification time
- PC Specification
- Comments

Each proof has its own page for structural, statistical and proof checking results with the attributes as follows:

- Theorem name
- Theorem conclusion
- Packagename
- Constructive proof (or not)
- Axioms
- Constructive lemmas (if any)
- Classical lemmas (if any)
- Package
- Comments

# Bibliography

- [1] Mark Adams. Introducing HOL zero. In *Mathematical Software–ICMS 2010*, pages 142–143. Springer, 2010.
- [2] Peter B Andrews. An introduction to mathematical logic and type theory: To truth through proof. 1986.
- [3] Andrew W Appel. Foundational proof-carrying code. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 247–256. IEEE, 2001.
- [4] Ali Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École Polytechnique, 2015.
- [5] Ali Assaf and Guillaume Burel. Translating HOL to Dedukti. In *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, pages 74–88, 2015.
- [6] Ali Assaf and Raphaël Cauderlier. Mixing HOL and Coq in Dedukti (Extended Abstract). In *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, pages 89–96, 2015.
- [7] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types*. Cambridge University Press, 2013.
- [8] Michael J Beeson. Constructive set theories. In *Foundations of Constructive Mathematics*, pages 162–201. Springer, 1985.
- [9] Chad E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, pages 111–117, 2012.
- [10] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 102–117, 2007.
- [11] Gilles Dowek. *Proofs and algorithms: an introduction to logic and computability*. Springer Science & Business Media, 2011.
- [12] Gilles Dowek. On the definition of the classical connectives and quantifiers. *Why is this a Proof?*, *Festschrift for Luiz Carlos Pereira*, 2015.
- [13] William M Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.



- [14] Gilda Ferreira and Paulo Oliva. On the relation between various negative translations. *Logic, Construction, Computation, Ontos-Verlag Mathematical Logic Series*, 3(23):227–258, 2012.
- [15] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In *Intelligent Computer Mathematics*, pages 267–281. Springer, 2014.
- [16] Frédéric Gilbert. A lightweight double-negation translation. In *LPAR-20. 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 2015.
- [17] Kurt Gödel. Zur intuitionistischen arithmetik und zahlentheorie. *Ergebnisse eines mathematischen Kolloquiums*, 4(1933):34–38, 1933.
- [18] Thomas C Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. In *The Kepler Conjecture*, pages 341–376. Springer, 2011.
- [19] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM (JACM)*, 40(1):143–184, 1993.
- [20] John Harrison. HOL light: An overview. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, pages 60–66, 2009.
- [21] Joe Hurd. The opentheory standard theory library. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, pages 177–191, 2011.
- [22] Hajime Ishihara. A note on the Gödel-Gentzen translation. *Mathematical Logic Quarterly*, 46(1):135–137, 2000.
- [23] Cezary Kaliszyk and Alexander Krauss. Scalable lcf-style proof translation. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 51–66, 2013.
- [24] Andrei Nikolaevich Kolmogorov. On the principle of excluded middle. *Mat. Sb*, 32(646-667):24, 1925.
- [25] Sigekatu Kuroda et al. Intuitionistische untersuchungen der formalistischen logik. *Nagoya Mathematical Journal*, 2:35–47, 1951.
- [26] Dale Miller. Unification of simply typed lambda-terms as logic programming. 1991.
- [27] Magnus O Myreen, Scott Owens, and Ramana Kumar. Steps towards verified implementations of HOL Light. In *Interactive Theorem Proving*, pages 490–495. Springer, 2013.
- [28] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.
- [29] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. *Unifying Theories in ProofPower-Z*, pages 123–140. Springer, 2006.

- [30] Florian Rabe. Representing isabelle in LF. In Karl Crary and Marino Miculan, editors, *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010*, volume 34 of *EPTCS*, pages 85–99, 2010.
- [31] Ronan Saillard. Dedukti: a universal proof checker. In *Foundation of Mathematics for Computer-Aided Formalization Workshop, January 9-11 2013, Padova, Italy*, 2013.
- [32] Carsten Schürmann and Mark-Oliver Stehr. An executable formalization of the HOL/Nuprl connection in the metalogical framework Twelf. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 150–166. Springer, 2006.
- [33] Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, pages 28–32, 2008.
- [34] Nik Sultana, Jasmin Christian Blanchette, and Lawrence C Paulson. LEO-II and Satallax on the Sledgehammer test bench. *Journal of Applied Logic*, 11(1):91–102, 2013.