HAL
open science

Translating HOL to Dedukti

Ali Assaf, Guillaume Burel

To cite this version:

Ali Assaf, Guillaume Burel. Translating HOL to Dedukti. 2014. hal-01097412v1

HAL Id: hal-01097412

https://hal.science/hal-01097412v1

Preprint submitted on 19 Dec 2014 (v1), last revised 18 Dec 2015 (v2)

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.

Translating HOL to Dedukti

Ali Assaf[1,2] and Guillaume Burel[3]

[1] INRIA Paris-Rocquencourt, Paris, France
[2] École polytechnique, Palaiseau, France
[3] ENSIIE/Cédric, Évry, France

Abstract. Dedukti is a logical framework based on the $\lambda\Pi$-calculus modulo rewriting, which extends the $\lambda\Pi$-calculus with rewrite rules. In this paper, we show how to translate the proofs of a family of HOL proof assistants to Dedukti. The translation preserves binding, typing, and reduction. We implemented this translation in an automated tool and used it to successfully translate the OpenTheory standard library.

1 Introduction

Following the LF legacy [14], Dedukti is a logical framework for defining logics and expressing proofs in those logics [7]. It is based on the *$\lambda\Pi$-calculus modulo rewriting*, which extends the *$\lambda\Pi$-calculus* with rewrite rules. Cousineau and Dowek [9] showed that functional *pure type systems* (PTS), a large class of calculi that are at the basis of many proof systems, can be embedded in the $\lambda\Pi$-calculus modulo in a way that is complete and reduction-preserving. This led to propose Dedukti as a universal proof framework.

In this paper, we focus on translating the proofs of HOL to Dedukti. HOL refers to a family of theorem provers built on a common logical system known as *higher-order logic* or *simple type theory* [8]. It includes systems such as HOL Light, HOL4, and ProofPower-HOL. These systems are fairly popular and a large number of important mathematical results have been formalized in them [12,13,24].

Universal proof checking Using Dedukti as a logical framework serves two goals. First, in the short term, it serves as an alternative, independent proof checker, providing an additional layer of confidence over each system. The second, longer term goal, is interoperability. Proof systems are becoming increasingly important, both in the formalization of mathematics and in software engineering. However, they are usually developed separately, with very little interoperability in mind. As a result, it is currently very difficult to reuse a proof from one system in another one. Embedding these different systems in a single unified framework is the first step to bring them closer together, and opens the way for theory management systems [15,22] to combine their proofs in order to construct and verify larger theories.

The $\lambda\Pi$-calculus as a logical framework The $\lambda\Pi$-calculus, also known as LF, is a typed λ-calculus with dependent types. Through the *Curry–Howard correspondence*, it can express a wide variety of logics [14]. Several formalizations of HOL in LF have been proposed [2,23,21].

The main concept behind this correspondence is the "*propositions as types*" principle. Typically, we can define a context declaring the types, terms, and judgments of the original logic, in such a way that provability in the logic corresponds to type inhabitation in the context. For HOL, the signature would be:

$$
\begin{aligned}
&\mathsf{type} &&: \mathsf{Type}\\
&\mathsf{prop} &&: \mathsf{type}\\
&\mathsf{arrow} &&: \mathsf{type} \to \mathsf{type} \to \mathsf{type}\\
&\mathsf{term} &&: \mathsf{type} \to \mathsf{Type}\\
&\mathsf{lam} &&: (\mathsf{term}\,\alpha \to \mathsf{term}\,\beta) \to \mathsf{term}\,(\mathsf{arrow}\,\alpha\,\beta)\\
&\mathsf{app} &&: \mathsf{term}\,(\mathsf{arrow}\,\alpha\,\beta) \to \mathsf{term}\,\alpha \to \mathsf{term}\,\beta\\
&\mathsf{proof} &&: \mathsf{term}\,\mathsf{prop}\\
&\mathsf{rule_1} &&: \ldots\\
&\mathsf{rule_2} &&: \ldots
\end{aligned}
$$

For each proposition ϕ of the logic, we can assign a type $\|\phi\|$ in the $\lambda\Pi$-calculus. The provability of the proposition ϕ corresponds to the inhabitation of the type $\|\phi\|$. Similarly, we can translate proofs as terms inhabiting those types, and the correctness of the proof corresponds to the well-typedness of the term.

However, because the $\lambda\Pi$-calculus does not have polymorphism, we cannot translate propositions directly as types, as doing so would prevent us from quantifying over propositions for example. Instead, for each proposition ϕ, we have two translations: one translation $|\phi|$ *as a term*, and another $\|\phi\| = \mathsf{proof}\,|\phi|$ *as a type*. This correspondence has been successfully used to embed logics in the LF framework [14,11], implemented in Twelf [20].

The $\lambda\Pi$-calculus vs. the $\lambda\Pi$-calculus modulo rewriting An important limitation of LF is that these encodings do not preserve reductions. For example, the term $(\lambda x : \alpha.\, x)\, x$ is encoded as $\mathsf{app}\,(\mathsf{lam}\,(\lambda x : \mathsf{term}\,\alpha.\, x))\; x$ which is not convertible to x. This is problematic not only because it makes the representation larger and hence inefficient but also because conversion proofs may be very long.

By extending the $\lambda\Pi$-calculus with rewrite rules such as

$$\mathsf{term}\,(\mathsf{arrow}\,\alpha\,\beta) \rightsquigarrow \mathsf{term}\,\alpha \to \mathsf{term}\,\beta\ ,$$

we can identify the type $\mathsf{term}\,(\mathsf{arrow}\,\alpha\,\beta)$ with the type $\mathsf{term}\,\alpha \to \mathsf{term}\,\beta$ and thus define a translation that is lighter and that preserve reductions. The encoding of the terms becomes more compact, as we represent λ-abstractions by λ-abstractions, applications by applications, etc. For example, the term $(\lambda x : \alpha.\, x)\, x$ is encoded as $(\lambda x : \mathsf{term}\,\alpha.\, x)\; x$. Such an encoding is impossible in LF for higher-order theories such as system F, HOL, or the calculus of constructions.

Moreover, our translation is modular enough so that we can extend the notion of reduction to the proofs of HOL and recover the pure type system nature of HOL [4]. This might be beneficial for several reasons:

1. It gives a reduction semantics for the proofs of HOL.
2. It allows compressing the proofs further by replacing conversion proofs with reflectivity.
3. Several other proof systems (Coq, Agda, etc.) are based on pure type systems, so expressing HOL as a PTS fits in the large scale of interoperability.

HOL and OpenTheory The theorem provers of the HOL family (HOL Light, HOL4, ProofPower-HOL, etc.) are built on a common logical formalism known as *higher-order logic*, and have fairly similar core implementations.

A recurrent issue when trying to retrieve proofs from these systems is that they do not keep a trace of their proofs [15,17,19]. Following the LCF architecture, they represent their theorems using an abstract datatype and thus guarantee their safety without the need to remember their proofs. This approach reduces memory consumption but hinders their ability to share proofs.

Fortunately, several proposals have already been made to solve this problem [15,19]. Among them is the OpenTheory project. It defines a standard format called the *article format* for recording and sharing HOL theorems. An article file contains a sequence of elementary commands to reconstruct the proofs. Importing a theorem requires only a mechanical execution of the commands.

The format is limited to the HOL family, and cannot be used to communicate the proofs of Coq for example. However, it is an excellent starting point for our translation. Choosing OpenTheory as a front-end has several advantages:

- We cover all the systems of the HOL family that can export their proofs to OpenTheory with a single implementation. As of today, this includes HOL Light, HOL4, and ProofPower-HOL.[4]
- The implementation of a theorem prover can change, so the existence of this standard, documented proof format is extremely helpful, if not necessary.
- The OpenTheory project also defines a large common standard theory library, covering the development of common datatypes and mathematical theories such as lists and natural numbers. This substantial body of theories was used as a benchmark for our implementation.

Related work Several formalizations of HOL in LF have been proposed [2,21,23]. To our knowledge, they lack an actual implementation of the translation. Other translations have been proposed to automatically extract the proofs of HOL to other systems such as Isabelle/HOL [16,19], Nuprl [18], or Coq [17]. With the exception of [16], these tools suffer from scalability problems. Our translation is lightweight and modular enough to be scalable and provides promising results. The implementation of Kalyszyk and Krauss [16] is the first efficient and scalable

[4] Isabelle/HOL can currently read from but not write to OpenTheory.

translation of HOL Light proofs, but its target is Isabelle/HOL, a system that is foundationally very close to HOL Light.

A project parallel to ours is Coqine [6], which proposes a translation of the *calculus of inductive constructions* (CIC), the formalism behind Coq, to Dedukti. The translation has been implemented in an automated tool that translates the proofs compiled by Coq to Dedukti. It can handle most of the features of Coq, and has been used to translate a part of its standard library.

Contributions We define a translation of the types, terms and proofs of HOL to Dedukti. We use the rewriting techniques of Cousineau and Dowek [9] to obtain a shallow embedding that is lightweight and modular. We implemented this translation in an automated tool called Holide, which automatically translates the proofs of HOL written in the OpenTheory format to Dedukti. We used it to successfully translate the OpenTheory standard library.

Outline The rest of this paper is organized as follows. Section 2 presents Dedukti and the $\lambda\Pi$-calculus modulo rewriting. Section 3 presents HOL and the logical system behind it. Section 4 defines the translation of HOL to Dedukti. We show that the translation is correct in Section 5. Section 6 discusses the details of our implementation and the results obtained by translating the OpenTheory standard library. Section 7 discusses some additional applications of rewriting. Finally, Section 8 summarizes and considers future work.

2 Dedukti

Dedukti is essentially a type checker for the $\lambda\Pi$-calculus modulo rewriting [7], which extends the $\lambda\Pi$-calculus with rewrite rules. We choose a presentation based on pure type systems [4], which makes no syntactic distinction between terms, usually denoted by M or N, and types, usually denoted by A or B.

We assume countably infinite sets of variables and constants. There are two sorts, Type and Kind. The sort Type is the type of types and the sort Kind is the type of Type. We write $\lambda x : A.\, M$ for abstractions and $M\, N$ for applications. The type of functions is written $\Pi x : A.\, B$, or $A \to B$ when x does not appear free in B. Application is left-associative while the arrow $\to$ is right-associative. Terms are considered up to α-equivalence. Contexts contain the type of variables while signatures contain the type of constants and their rewrite rules.

Definition 1. *The syntax of the $\lambda\Pi$-calculus modulo rewriting is:*

$$
\begin{array}{lll}
\textit{variables} & x, y & \\
\textit{constants} & c & \\
\textit{sorts} & s & ::= \mathsf{Type} \mid \mathsf{Kind} \\
\textit{terms} & M, N, A, B & ::= x \mid c \mid s \mid \Pi x : A.\, B \mid \lambda x : A.\, M \mid M\, N \\
\textit{contexts} & \Gamma, \Delta & ::= \cdot \mid \Gamma, x : A \\
\textit{signatures} & \Sigma & ::= \cdot \mid \Sigma, c : A \mid \Sigma, [\Gamma]\ M \rightsquigarrow N
\end{array}
$$

If R is a set of rewrite rules, we write $\longrightarrow_R$ for the reduction relation, $\longrightarrow_R^+$ for its transitive closure, $\longrightarrow_R^*$ for its reflexive transitive closure, and $\equiv_R$ for its reflexive symmetric transitive closure. Given a signature Σ, we write $\beta\Sigma$ for the union of the β rule with the rewrite rules of Σ.

The typing judgments $\Sigma \mid \Gamma \vdash M : A$ are accompanied by context formation judgments $\Sigma \mid \Gamma$ *context* and signature formation judgments Σ *signature*. We write $\Gamma \vdash M : A$ and Γ *context* instead of $\Sigma \mid \Gamma \vdash M : A$ and $\Sigma \mid \Gamma$ *context* when the signature is not ambiguous. The rules are presented in Figure 1.

$$\frac{\Gamma \ \mathit{context} \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Var} \qquad \frac{\Gamma \ \mathit{context} \qquad (c : A) \in \Sigma}{\Gamma \vdash c : A}\ \textsc{Const}$$

$$\frac{\Gamma \ \mathit{context}}{\Gamma \vdash \mathsf{Type} : \mathsf{Kind}}\ \textsc{Type} \qquad \frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}\ \textsc{Prod}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}\ \textsc{Abs} \qquad \frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : [N/x]B}\ \textsc{App}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \mathsf{Type} \qquad A \equiv_{\beta\Sigma} B}{\Gamma \vdash M : B}\ \textsc{Conv}$$

$$\frac{\Sigma \ \mathit{signature}}{\cdot \ \mathit{context}}\ \textsc{EmptyCtx} \qquad \frac{\Gamma \vdash A : \mathsf{Type} \qquad x \notin \Gamma}{\Gamma, x : A \ \mathit{context}}\ \textsc{VarCtx}$$

$$\frac{}{\cdot \ \mathit{signature}}\ \textsc{EmptySig} \qquad \frac{\Sigma \mid \cdot \vdash A : s \qquad c \notin \Sigma}{\Sigma, c : A \ \mathit{signature}}\ \textsc{ConstSig}$$

$$\frac{\Sigma \mid \Gamma \vdash M : A \qquad \Sigma \mid \Gamma \vdash N : A}{\Sigma, [\Gamma]\ M \rightsquigarrow N \ \mathit{signature}}\ \textsc{RewriteSig}$$

Fig. 1. Typing rules of the $\lambda\Pi$-calculus

Example 1. Let Σ be the signature containing

$$\mathsf{nat} : \mathsf{Type},\ \mathsf{z} : \mathsf{nat},\ \mathsf{s} : \mathsf{nat} \to \mathsf{nat},\ \mathsf{stream} : \mathsf{Type}$$

and the rewrite rule

$$[\cdot]\ \mathsf{stream} \rightsquigarrow \mathsf{nat} \to \mathsf{nat}\ .$$

The term $\lambda f : \mathsf{stream}.\, \lambda x : \mathsf{nat}.\, \mathsf{s}\,(f\,x)$ is well-typed in Σ and has the type $\mathsf{stream} \to \mathsf{stream}$. Notice that this term would not be well-typed without the rewrite rule.

Dedukti imposes some additional restrictions on the rewrite rules to keep type-checking decidable. In particular, the left side of a rewrite rule must belong to the pattern fragment and the free variables of the right side must appear on the left side. Moreover, the reduction relation $\longrightarrow_{\beta\Sigma}$ should be confluent and strongly normalizing. This property is not verified by the system and it is up to the user to ensure that it is indeed the case. We will do so in Section 5.

3 HOL

There are many different formulations for higher-order logic. The intuitionistic formulation is based on implication and universal quantification as primitive connectives, but the current systems generally use a formulation called Q_0 [1] based on equality as a primitive connective. We take as reference the logical system used by OpenTheory [15], which we will now briefly present.

The terms of the logic are terms of the simply typed λ-calculus, with a base type bool representing the type of propositions and a type ind of individuals. The terms can contain constant symbols such as $(=)$, the symbol for equality, or select, the symbol of choice. The logic supports a restricted form of polymorphism, known as ML-style polymorphism, by allowing type variables, such as α or β, to appear in types. For example, the type of $(=)$ is $\alpha \to \alpha \to \mathsf{bool}$.

Types can be parameterized through type operators of the form $p(A_1, \ldots, A_n)$. For example, list is a type operator of arity 1, and $\mathsf{list}(\mathsf{bool})$ is the type of lists of booleans. Type variables and type operators are enough to describe all the types of HOL, because bool can be seen as a type operator of arity 0, and the arrow $\to$ as a type operator of arity 2. Hence the type of $(=_\alpha)$ is in fact $\to(\alpha, \to(\alpha, \mathsf{bool}()))$. We still write $A \to B$ instead of $\to(\mathsf{A}, \mathsf{B})$ for arrow types, p instead of $p()$ for type operators of arity 0, and $M = N$ instead of $(=)\, M\, N$ when it is more convenient.

Definition 2. *The syntax of HOL is:*

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

The propositions of the logic are the terms of type bool and the predicates are the terms of type $A \to \mathsf{bool}$. We use letters such as ϕ or ψ to denote propositions. The contexts, denoted by Γ or Δ, are sets of propositions, and the judgments of the logic are of the form $\Gamma \vdash \phi$. The derivation rules are presented in Figure 2.

$$\frac{}{\vdash M = M}\ \textsc{Refl}\ M \qquad \frac{\Gamma \vdash M = N}{\Gamma \vdash \lambda x : A.\, M = \lambda x : A.\, N}\ \textsc{AbsThm}\ x$$

$$\frac{\Gamma \vdash F = G \qquad \Delta \vdash M = N}{\Gamma \cup \Delta \vdash F\, M = G\, N}\ \textsc{AppThm} \qquad \frac{}{\vdash (\lambda x : A.\, M)\, x = M}\ \textsc{Beta}\ x\ M$$

$$\frac{}{\{\phi\} \vdash \phi}\ \textsc{Assume} \qquad \frac{\Gamma \vdash \phi = \psi \qquad \Delta \vdash \phi}{\Gamma \cup \Delta \vdash \psi}\ \textsc{EqMp}$$

$$\frac{\Gamma \vdash \phi \qquad \Delta \vdash \psi}{(\Gamma - \{\psi\}) \cup (\Delta - \{\phi\}) \vdash \phi = \psi}\ \textsc{DeductAntiSym} \qquad \frac{\Gamma \vdash \phi}{\Gamma[\sigma] \vdash \phi[\sigma]}\ \textsc{Subst}\ \sigma$$

Fig. 2. Derivation rules of HOL

Example 2. Here is a derivation of the transitivity of equality: if $\Gamma \vdash x = y$ and $\Delta \vdash y = z$, then $\Gamma \cup \Delta \vdash x = z$.

$$\frac{\dfrac{\dfrac{}{\vdash ((=)\, x) = ((=)\, x)}\ \textsc{Refl} \qquad \Delta \vdash y = z}{\Delta \vdash (x = y) = (x = z)}\ \textsc{AppThm} \qquad \Gamma \vdash x = y}{\Gamma \cup \Delta \vdash x = z}\ \textsc{EqMp}$$

HOL supports mechanisms for defining new types and constants in a conservative way. We will not consider them here, as their treatment should be straightforward. In addition to the core derivation rules, three axioms are assumed:

- η-equality, which states that $\lambda x : A.\ M\, x = M$,
- the axiom of choice, with a predeclared symbol of choice called select,
- the axiom of infinity, which states that the type ind is infinite.

It is important to note that from η-convertibility and the axiom of choice, we can derive the excluded middle [5], making HOL a classical logic.

4 Translation

In this section we show how to translate HOL to Dedukti. We define a signature Σ which contains the primitive declarations and definitions, and a translation function assigning, to every construct of the logic, a term that is well-typed in the signature Σ.

HOL Types To translate the simple types of HOL, we declare a new Dedukti type called type and three constructors bool, ind and arrow.

$$\begin{array}{ll} \mathsf{type} & : \mathsf{Type} \\ \mathsf{bool} & : \mathsf{type} \\ \mathsf{ind} & : \mathsf{type} \\ \mathsf{arrow} & : \mathsf{type} \to \mathsf{type} \to \mathsf{type} \end{array}$$

One should not confuse type, which represents HOL types, with Type, the sort of types of the $\lambda\Pi$-calculus modulo. The translation as a term is defined inductively on the structure of the simple type.

Definition 3. *For any HOL type A, we define $|A|$, the translation of A as a term, to be*

$$\begin{array}{ll} |\alpha| & = \alpha \\ |\mathsf{bool}| & = \mathsf{bool} \\ |\mathsf{ind}| & = \mathsf{ind} \\ |A \to B| & = \mathsf{arrow}\,|A|\;|B| \;\; . \end{array}$$

More generally, if we have an n-ary HOL type operator p, we declare a constant p of type $\underbrace{\mathsf{type} \to \ldots \to \mathsf{type}}_{n} \to \mathsf{type}$, and we translate an instance $p\,(A_1, \ldots, A_n)$ of this type operator to the term $p\,|A_1| \cdots |A_n|$.

HOL Terms We declare a new dependent type called term indexed by type, and we identify the terms of type $\mathsf{term}(\mathsf{arrow}\,A\,B)$ with the functions of type $\mathsf{term}\,A \to \mathsf{term}\,B$ by adding a rewrite rule. We also declare a constant eq for HOL equality and a constant select for the choice operator.

$$\begin{array}{ll} \mathsf{term} & : \mathsf{type} \to \mathsf{Type} \\ \mathsf{eq} & : \Pi\alpha : \mathsf{type}.\ \mathsf{term}\,(\mathsf{arrow}\,\alpha\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})) \\ \mathsf{select} & : \Pi\alpha : \mathsf{type}.\ \mathsf{term}\,(\mathsf{arrow}\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})\ \alpha) \end{array}$$

$$[\alpha : \mathsf{type}, \beta : \mathsf{type}]\ \mathsf{term}\,(\mathsf{arrow}\,\alpha\,\beta) \rightsquigarrow \mathsf{term}\,\alpha \to \mathsf{term}\,\beta$$

The symbol term can be seen as a decoding function that assigns a Dedukti type to every HOL type. The translation of a term M of type A will then be a term of type $\mathsf{term}\,|A|$.

Definition 4. *For any HOL type A, we define $\|A\| = \mathsf{term}\,|A|$.*

Definition 5. *For any HOL term M, we define $|M|$, the translation of M as a term to be*

$$\begin{array}{ll} |x| & = x \\ |M\,N| & = |M|\;|N| \\ |\lambda x : A.\,M| & = \lambda x : \|A\|\,.\,|M| \\ |(=_A)| & = \mathsf{eq}\,|A| \\ |\mathsf{select}_A| & = \mathsf{select}\,|A| \;\; . \end{array}$$

More generally, for every HOL constant c of type A, if $\alpha_1, \ldots, \alpha_n$ are the free type variables that appear in A, we declare a new constant c of type

$$\Pi \alpha_1 : \mathsf{type}. \ldots \Pi \alpha_n : \mathsf{type}. \, \|A\|$$

and we translate an instance $c_{A_1,\ldots,A_n}$ of this constant by the term $c\,|A_1| \cdots |A_n|$.

Example 3. The term $(\lambda x : \alpha.\, x)\, x$ is translated to

$$|(\lambda x : \alpha.\, x)\, x| = (\lambda x : \mathsf{term}\, \alpha.\, x)\, x$$

which is convertible to x.

HOL Proofs We declare a new type proof, to express the proof judgments of HOL. It is a dependent type, indexed by the proposition it is proving, and corresponds to the judgment $\Gamma \vdash \phi$.

$$\mathsf{proof} : \mathsf{term}\, \mathsf{bool} \to \mathsf{Type}$$

Definition 6. *For any HOL proposition ϕ, we define $\|\phi\| = \mathsf{proof}\, |\phi|$. For any HOL context $\Gamma = \phi_1, \ldots, \phi_n$, we define*

$$\|\Gamma\| = h_{\phi_1} : \|\phi_1\|, \ldots, h_{\phi_n} : \|\phi_n\|$$

where $h_{\phi_1}, \ldots, h_{\phi_n}$ are fresh variables.

We now take care of the derivation rules of HOL (Figure 2). In the following, we write $\Pi x, y : A.\, B$ as a shortcut for $\Pi x : A.\, \Pi y : A.\, B$.

Equality proofs We declare Refl, FunExt, and AppThm:

$$\begin{array}{ll}
\mathsf{Refl} & : \Pi \alpha : \mathsf{type}.\, \Pi x : \mathsf{term}\, \alpha.\ \mathsf{proof}\, (\mathsf{eq}\, \alpha\, x\, x) \\
\mathsf{FunExt} & : \Pi \alpha, \beta : \mathsf{type}.\, \Pi f, g : \mathsf{term}\, (\mathsf{arrow}\, \alpha\, \beta)\,. \\
& \quad (\Pi x : \mathsf{term}\, \alpha.\ \mathsf{proof}\, (\mathsf{eq}\, \beta\, (f\, x)\ (g\, x))) \to \mathsf{proof}\, (\mathsf{eq}\, (\mathsf{arrow}\, \alpha\, \beta)\ f\, g) \\
\mathsf{AppThm} & : \Pi \alpha, \beta : \mathsf{type}.\, \Pi f, g : \mathsf{term}\, (\mathsf{arrow}\, \alpha\, \beta)\,.\, \Pi x, y : \mathsf{term}\, \alpha. \\
& \quad \mathsf{proof}\, (\mathsf{eq}\, (\mathsf{arrow}\, \alpha\, \beta)\ f\, g) \to \mathsf{proof}\, (\mathsf{eq}\, \alpha\, x\, y) \to \mathsf{proof}\, (\mathsf{eq}\, \beta\, (f\, x)\ (g\, y))
\end{array}$$

The constant FunExt corresponds to *functional extensionality*, which states that if two functions f and g of type $A \to B$ are equal on all values x of type A, then f and g are equal. We can use it to translate both the AbsThm rule and the η axiom. Since our encoding is shallow, β-equality can be proved by reflexivity.

Definition 7. *The rules* Refl, AbsThm, AppThm, *and* Beta *are translated to*

$$\left| \frac{}{\vdash M = M}\ \textsc{Refl} \right| = \mathsf{Refl}\, |A|\ |M| \quad \textit{(where } A \textit{ is the type of } M\textit{)}$$

$$\left|\frac{\mathcal{D}}{\Gamma \vdash \lambda x : A.\, M = \lambda x : A.\, N}\ \textsc{AbsThm}\right| =$$

$$\mathsf{FunExt}\,|A|\ |B|\ |\lambda x : A.\, M|\ |\lambda x : A.\, N|\ (\lambda x : |A|\,.\ |\mathcal{D}|)$$

$$\left|\frac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma \cup \Delta \vdash F\, M = G\, N}\ \textsc{AppThm}\right| = \mathsf{AppThm}\,|A|\ |B|\ |F|\ |G|\ |M|\ |N|\ |\mathcal{D}_1|\ |\mathcal{D}_2|$$

$$\left|\frac{}{(\lambda x : A.\, M)\, x = M}\ \textsc{Beta}\right| = \mathsf{Refl}\,|A|\ |M| \quad \textit{(where } A \textit{ is the type of } M\textit{)}\ .$$

Boolean proofs We declare the constants PropExt and EqMp:

$$\begin{aligned}
&\mathsf{PropExt} : \Pi p, q : \mathsf{term\ bool}.\\
&\qquad (\mathsf{proof}\ q \to \mathsf{proof}\ p) \to (\mathsf{proof}\ q \to \mathsf{proof}\ p) \to \mathsf{proof}\ (\mathsf{eq\ bool}\ p\ q)\\
&\mathsf{EqMp} \quad : \Pi p, q : \mathsf{term\ bool}.\ \mathsf{proof}\ (\mathsf{eq\ bool}\ p\ q) \to \mathsf{proof}\ p \to \mathsf{proof}\ q
\end{aligned}$$

The constant PropExt corresponds to *propositional extensionality* and, together with EqMp, states that equality on booleans in HOL behaves like the connective "*if and only if*".

Definition 8. *The rules* Assume, DeductAntiSym, *and* EqMp *are translated to*

$$\left|\frac{}{\{\phi\} \vdash \phi}\ \textsc{Assume}\right| = h_\phi \quad \textit{(where } h_\phi \textit{ is a fresh variable)}$$

$$\left|\frac{\mathcal{D}_1 \qquad \mathcal{D}_2}{(\Gamma - \{\psi\}) \cup (\Delta - \{\phi\}) \vdash \phi = \psi}\ \textsc{DeductAntiSym}\right| =$$

$$\mathsf{PropExt}\,|\phi|\ |\psi|\ (\lambda h_\psi : \|\psi\|\,.\ |\mathcal{D}_1|)\ (\lambda h_\phi : \|\phi\|\,.\ |\mathcal{D}_2|)$$

$$\left|\frac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma \cup \Delta \vdash \psi}\ \textsc{EqMp}\right| = \mathsf{EqMp}\,|\phi|\ |\psi|\ |\mathcal{D}_1|\ |\mathcal{D}_2|\ .$$

Substitution proofs The HOL rule SUBST derives $\Gamma[\sigma] \vdash \phi[\sigma]$ from $\Gamma \vdash \phi$. In OpenTheory, the substitution can substitute for both term and type variables but type variables are instantiated first. For the sake of clarity, we split this rule in two steps: one for term substitution of the form $\sigma = M_1/x_1, \ldots, M_n/x_n$, and one for type substitution of the form $\theta = A_1/\alpha_1, \ldots, A_m/\alpha_m$. In Dedukti, we have to rely on β-reduction to express substitution. We can correctly translate a parallel substitution $M[M_1/x_1, \ldots, M_n/x_n]$ as

$$(\lambda x_1 : B_1. \ldots \lambda x_n : B_n.\, M)\, M_1 \ldots M_n$$

where B_i is the type of M_i.

Definition 9. *The rule* SUBST *is translated to*

$$\left|\frac{\mathcal{D}}{\Gamma[\theta] \vdash \phi[\theta]}\ \textsc{TypeSubst}\right| = (\lambda\alpha_1 : \mathsf{type}.\ \ldots \lambda\alpha_m : \mathsf{type}.\ |\mathcal{D}|)\ |A_1|\ \ldots\ |A_m|$$

$$\left|\frac{\mathcal{D}}{\Gamma[\sigma] \vdash \phi[\sigma]}\ \textsc{TermSubst}\right| = (\lambda x_1 : \|B_1\|.\ \ldots \lambda x_n : \|B_n\|.\ |\mathcal{D}|)\ |M_1|\ \ldots\ |M_n|$$

5 Correctness

The correctness of the translation is expressed by two properties: *completeness* and *soundness*. The first states that all the generated terms have the correct type. For example, the translation of a term of type A has type $\|A\|$ while the translation of a proof of ϕ has type $\|\phi\|$. The second states that if a proof term is well-typed in Dedukti, then the proof is correct in the original logic. These two properties ensure that we can use Dedukti as an independent proof checker.

Completeness Let Σ be the signature of HOL containing the declarations and rewrite rules of the previous sections.

Lemma 1. *For any HOL type A,*

$$\Sigma \mid \alpha_1 : \mathsf{type}, \ldots, \alpha_n : \mathsf{type} \vdash |A| : \mathsf{type}$$

where $\alpha_1, \ldots, \alpha_n$ are the free type variables appearing in A.

Lemma 2. *For any HOL term M of type A,*

$$\Sigma \mid \alpha_1 : \mathsf{type}, \ldots, \alpha_n : \mathsf{type}, x_1 : \|A_1\|, \ldots x_n : \|A_n\| \vdash |M| : \|A\|$$

where $\alpha_1, \ldots, \alpha_n$ are the free type variables and $x_1 : A_1, \ldots, x_n : A_n$ are the free term variables appearing in M.

Theorem 1. *For any HOL proof $\mathcal{D}$ of $\Gamma \vdash \phi$,*

$$\Sigma \mid \alpha_1 : \mathsf{type}, \ldots, \alpha_n : \mathsf{type}, x_1 : \|A_1\|, \ldots x_n : \|A_n\|, \|\Gamma\| \vdash |\mathcal{D}| : \|\phi\|$$

where $\alpha_1, \ldots, \alpha_n$ are the free type variables and $x_1 : A_1, \ldots, x_n : A_n$ are the free term variables appearing in $\mathcal{D}$.

Proof. By induction on the structure of $\mathcal{D}$.

Soundness Proving soundness is less straightforward than proving completeness. In fact, the soundness of the embedding is closely related to the confluence and normalization properties of the system. We just state the results here and refer the reader to [3,9,10] for the complete proofs.[5]

Lemma 3. *The reduction relation $\longrightarrow_{\beta\Sigma}$ is confluent.*

Lemma 4. *The reduction relation $\longrightarrow_{\beta\Sigma}$ is strongly normalizing.*

Theorem 2. *If $\Sigma \mid \|\Gamma\| \vdash |\mathcal{D}| : \|A\|$ then $\mathcal{D}$ is a correct proof of A in HOL.*

[5] The terms *soundness* and *completeness* are interchanged in paper [9].

Package	Size (KB)		Time (s)	
	OpenTheory	Dedukti	Translation	Verification
unit	26	114	0.2	0.0
function	89	581	0.3	0.2
pair	216	1435	0.8	0.5
bool	305	1782	0.9	0.5
sum	501	3667	2.1	1.1
option	551	3933	2.2	1.2
relation	965	7923	4.6	2.8
list	1444	9891	5.7	3.2
real	1752	11402	6.5	3.1
natural	2112	12248	6.8	3.2
set	2343	17632	10.2	5.8
Total	10304	70608	40.3	21.6

Table 1. Translation of the OpenTheory standard library

6 Implementation

We implemented this translation in an automated tool called Holide[6]. The program works as an implementation of the OpenTheory virtual machine that keeps track of the corresponding proof terms for the theorems. HOL proofs are known to be very large [16,17,19], and we needed to implement proof sharing to reduce them to a manageable size. OpenTheory already provides some form of proof sharing but we found that it was easier to completely factorize the derivations into individual intermediary steps.

Holide reads a HOL proof written in the OpenTheory article format (`.art`) and outputs a Dedukti file (`.dk`) containing its translation. We can run Dedukti on the generated file to verify it. All generated files are linked with a predefined file `hol.dk` containing the signature Σ that we defined in Section 4.

We used Holide to translate the OpenTheory standard library. The library is organized into logical packages, each corresponding to a coherent theory such as lists or natural numbers. We were able to verify all of the generated files. The results are summarized in Table 1. We list the size of the files generated by the translation, as well as the time it takes to translate and verify each package. These tests were done on a 64-bit Intel Xeon(R) CPU @ 2.67GHz $\times$ 4 machine with 4 GB of RAM.

7 Extensions

In this section we show some additional advantages of having a translation which preserves reductions.

[6] The software available at https://www.rocq.inria.fr/deducteam/Holide/.

Compressing conversion proofs One of the reasons why HOL proofs are so large is that conversion proofs have to traverse the terms using the congruence rules AbsThm and AppThm. Since we now prove β-reduction using reflexivity, large conversion proofs could be reduced to a single reflexivity step, therefore reducing the size of the proofs.[7]

Example 4. The following proof of $f(g((\lambda x : A.\, x)\, x)) = f(g(x))$,

$$\dfrac{\dfrac{}{\vdash f = f}\,\textsc{Refl}\ f \qquad \dfrac{\dfrac{}{\vdash g = g}\,\textsc{Refl}\ g \qquad \dfrac{}{\vdash (\lambda x : A.\, x)\, x = x}\,\textsc{Beta}}{\vdash g((\lambda x : A.\, x)\, x) = g\, x}\,\textsc{AppThm}}{\vdash f(g((\lambda x : A.\, x)\, x)) = f(g\, x)}$$

can be translated simply as $\mathsf{Refl}\, B(f(g\, x))$.

HOL as a pure type system It turns out that HOL can be seen as a pure type system called λ_{HOL} with three sorts [4]. This formulation corresponds to intuitionistic higher-order logic. However, this structure is lost in the Q_0 formulation used by the HOL systems. Our shallow embedding can be adapted to recover this structure.

Instead of equality, we declare implication and universal quantification as primitive connectives, and we define what provability means through rewriting.

$$\begin{aligned}
&\mathsf{imp} \ : \mathsf{term}\,(\mathsf{arrow}\,\mathsf{bool}\,(\mathsf{arrow}\,\mathsf{bool}\,\mathsf{bool}))\\
&\mathsf{forall} : \Pi\alpha : \mathsf{type}.\ \mathsf{term}\,(\mathsf{arrow}\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})\ \mathsf{bool})
\end{aligned}$$

$$\begin{aligned}
&[p : \mathsf{term}\,\mathsf{bool}, q : \mathsf{term}\,\mathsf{bool}] && \mathsf{proof}\,(\mathsf{imp}\,p\,q) \rightsquigarrow \mathsf{proof}\,p \to \mathsf{proof}\,q\\
&[\alpha : \mathsf{type}, p : \mathsf{term}\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})] && \mathsf{proof}\,(\mathsf{forall}\,p) \rightsquigarrow \Pi x : \mathsf{term}\,\alpha.\ \mathsf{proof}\,(p\,x)
\end{aligned}$$

However, this time we do not need to declare constants like Refl and AppThm for the derivation rules, because they are derivable. Here is a derivation of the introduction and elimination rules for implication for example:

$$\begin{aligned}
\mathsf{imp_intro} \ &: \Pi p, q : \mathsf{term}\,\mathsf{bool}.\ (\mathsf{proof}\,p \to \mathsf{proof}\,q) \to \mathsf{proof}\,(\mathsf{imp}\,p\,q)\\
&= \lambda p, q : \mathsf{term}\,\mathsf{bool}.\ \lambda h : (\mathsf{proof}\,p \to \mathsf{proof}\,q)\,.\,h\\
\mathsf{imp_elim} \ &: \Pi p, q : \mathsf{term}\,\mathsf{bool}.\ \mathsf{proof}\,(\mathsf{imp}\,p\,q) \to \mathsf{proof}\,p \to \mathsf{proof}\,q\\
&= \lambda p, q : \mathsf{term}\,\mathsf{bool}.\ \lambda h : \mathsf{proof}\,(\mathsf{imp}\,p\,q)\,.\,\lambda x : \mathsf{proof}\,p.\ h\,x
\end{aligned}$$

By translating the introduction rules as λ-abstractions, and the elimination rules as applications, we recover the reduction of the proof terms, which corresponds to *cut elimination*.

As for equality, it is also possible to define it in terms of these connectives. For example, we could use the Leibniz definition of equality, which is the one used by Coq:

$$\begin{aligned}
\mathsf{eq} \ &: \Pi\alpha : \mathsf{type}.\ \mathsf{term}\,(\mathsf{arrow}\,\alpha\,(\mathsf{arrow}\,\alpha\,\mathsf{bool}))\\
&= \lambda\alpha : \mathsf{type}.\ \lambda x : \mathsf{term}\,\alpha.\ \lambda y : \mathsf{term}\,\alpha.\\
&\quad \mathsf{forall}\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})\ (\Pi p : \mathsf{term}\,(\mathsf{arrow}\,\alpha\,\mathsf{bool})\,.\ \mathsf{imp}\,(p\,x)\ (p\,y))
\end{aligned}$$

[7] This also applies to conversions involving constant definitions, which we did not cover here but are also assumed as an axiom in HOL.

We would still need to assume some axioms to prove all the rules, namely FunExt and PropExt [17], but this definition is closer to that of Coq. Since the λ_{HOL} PTS is a strict subset of the calculus of inductive constructions, we could imagine a translation that injects HOL directly into an embedding of Coq in Dedukti [6].

8 Conclusion

We showed how to translate HOL to Dedukti by adapting techniques from Cousineau and Dowek [9] to define an embedding that is sound, complete, and reduction preserving. Using our implementation, we were able to translate the OpenTheory standard library and verify it in Dedukti.

Future work The translation we have presented can be improved in several ways. The current implementation suffers from a lack of linking: if a package makes use of a type, constant, or theorem defined in another package, we do not have a reference to the original definition. This is due to a limitation of the OpenTheory article format. In OpenTheory, this problem is resolved by adding a theory management layer, which is responsible for composing and linking theories together [15]. It would be beneficial to integrate this layer in our translation so that we can properly link the resulting files together.

While we used several optimizations including term sharing in our implementation, there is still room for reducing the time and memory consumption of the translation and the size of the generated files. The caching techniques of Kaliszyk and Krauss [16] could be used in this regard to handle larger libraries and formalizations.

Finally, we can study how to combine the proofs obtained by this translation with the proofs obtained from the translation of Coq [6]. That will require a careful examination of the compatibility of the two embeddings. First, the types of the two theories must coincide, so that a natural number from HOL is the same as a natural number from Coq for example. Second, we must make sure that the resulting theory is consistent. For instance, we know that every type in HOL is inhabited, which is inconsistent with the existence of empty types in Coq, so we will need to modify the translations to avoid this. A solution is to parameterize each HOL type variable by a witness ensuring that it is non-empty. Our translation is modular enough to allow this solution without much trouble.

Acknowledgments We thank Gilles Dowek for his support, as well as Mathieu Boespflug and Chantal Keller for their comments and suggestions.

References

1. Andrews, P.B.: An introduction to mathematical logic and type theory: to truth through proof. Academic Press Professional, Inc., San Diego, CA, USA (1986)
2. Appel, A.W.: Foundational proof-carrying code. In: LICS. p. 247–256. IEEE Computer Society, Washington, DC, USA (2001)

3. Assaf, A.: Conservativity of embeddings in the lambda-Pi calculus modulo rewriting (2013), https://hal.archives-ouvertes.fr/hal-01084165, <hal-01084165>
4. Barendregt, H.P.: Lambda Calculi with Types, Handbook of Logic in Computer Science Vol. II. Oxford University Press (1992)
5. Beeson, M.: Foundations of Constructive Mathematics. Springer-Verlag (1985)
6. Boespflug, M., Burel, G.: CoqInE: translating the calculus of inductive constructions into the lambda-pi-calculus modulo. In: PxTP. pp. 44–50 (2012)
7. Boespflug, M., Carbonneaux, Q., Hermant, O.: The lambda-Pi-calculus modulo as a universal proof language. In: PxTP. pp. 28–43 (2012)
8. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5(02), 56–68 (Jun 1940)
9. Cousineau, D., Dowek, G.: Embedding pure type systems in the lambda-Pi-calculus modulo. In: Rocca, S.R.D. (ed.) TLCA, pp. 102–117. No. 4583 in LNCS, Springer Berlin Heidelberg (2007)
10. Dowek, G.: Models and termination of proof-reduction in the $\lambda\Pi$-calculus modulo theory (2014), https://who.rocq.inria.fr/Gilles.Dowek/Publi/superpi.pdf
11. Geuvers, H., Barendsen, E.: Some logical and syntactical observations concerning the first-order dependent type system lambda-P. Mathematical Structures in Computer Science 9(04), 335–359 (1999)
12. Hales, T.C.: The Jordan curve theorem, formally and informally. American Mathematical Monthly 114(10), 882–894 (2007)
13. Hales, T.C., Harrison, J., McLaughlin, S., Nipkow, T., Obua, S., Zumkeller, R.: A revision of the proof of the Kepler conjecture. In: Lagarias, J.C. (ed.) The Kepler Conjecture, pp. 341–376. Springer New York (2011)
14. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. J. ACM 40(1), 143–184 (1993)
15. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM, pp. 177–191. No. 6617 in LNCS, Springer (2011)
16. Kaliszyk, C., Krauss, A.: Scalable LCF-style proof translation. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP, pp. 51–66. No. 7998 in LNCS, Springer Berlin Heidelberg (2013)
17. Keller, C., Werner, B.: Importing HOL light into coq. In: Kaufmann, M., Paulson, L.C. (eds.) ITP, pp. 307–322. No. 6172 in LNCS, Springer Berlin Heidelberg (2010)
18. Naumov, P., Stehr, M.O., Meseguer, J.: The HOL/NuPRL proof translator. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs, pp. 329–345. No. 2152 in LNCS, Springer Berlin Heidelberg (2001)
19. Obua, S., Skalberg, S.: Importing HOL into Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning, pp. 298–302. No. 4130 in LNCS, Springer Berlin Heidelberg (2006)
20. Pfenning, F., Schürmann, C.: System description: Twelf — a meta-logical framework for deductive systems. In: CADE-16, pp. 202–206. No. 1632 in LNCS, Springer Berlin Heidelberg (1999)
21. Rabe, F.: Representing Isabelle in LF. EPTCS 34, 85–99 (Sep 2010), arXiv: 1009.2794
22. Rabe, F., Kohlhase, M.: A scalable module system. arXiv:1105.0548 (2011)
23. Schürmann, C., Stehr, M.O.: An executable formalization of the HOL/Nuprl connection in the metalogical framework Twelf. In: Hermann, M., Voronkov, A. (eds.) LPAR, pp. 150–166. No. 4246 in LNCS, Springer Berlin Heidelberg (2006)
24. Wiedijk, F.: The QED manifesto revisited. Studies in Logic, Grammar and Rhetoric 10(23), 121–133 (2007)