Conversion of HOL Light proofs into Metamath

MARIO M. CARNEIRO
Department of Mathematics, Ohio State University, Columbus OH 43210, USA

We present an algorithm for converting proofs from the OpenTheory interchange format, which can be translated to and from any of the HOL family of proof languages (HOL4, HOL Light, ProofPower, and Isabelle), into the ZFC-based Metamath language. This task is divided into two steps: the translation of an OpenTheory proof into a Metamath HOL formalization, hol.mm, followed by the embedding of the HOL formalization into the main ZFC foundations of the main Metamath library, set.mm. This process provides a means to link the simplicity of the Metamath foundations to the intense automation efforts which have borne fruit in HOL Light, allowing the production of complete Metamath proofs of theorems in HOL Light, while also proving that HOL Light is consistent, relative to Metamath’s ZFC axiomatization.

1. INTRODUCTION

Metamath is a proof language, developed in 1992 on the principle of minimizing the foundational logic to as little as possible [Meg07]. The resulting logic has only one built-in rule of inference, direct substitution, and all syntax and axioms are input outside the logical core. The most well-developed axiom system in Metamath is called set.mm, and adds the axioms of classical propositional calculus, first-order predicate calculus, and ZFC set theory to the Metamath foundations; this axiom system and associated library of theorems is also sometimes referred to as Metamath. In contrast, OpenTheory is an interchange format for the HOL family of proof languages (HOL4, HOL Light, ProofPower, and Isabelle) based on a higher-order logical kernel [Hur11]. All the axioms of HOL are built into the kernel, and several axioms perform proper substitution, which involves the renaming of bound variables, as part of their operation. The goal of this paper is to present an algorithm that transforms valid theorem derivations in OpenTheory into equivalent theorem derivations in set.mm.

The main task divides neatly into two parts. First, the axioms and inferences of OpenTheory are translated into their Metamath equivalents, producing a new database of axioms which we will call hol.mm. At this stage it is still essentially a HOL derivation, but the inferences are done “the Metamath way.” The primary job here is to eliminate proper substitutions and dummy variable renaming, which must be done over several steps in Metamath, and introduce metavariables in place of free variables in the final statement.

The second task is to convert a hol.mm derivation into a set.mm derivation. This
step is done entirely within Metamath, and works by constructing a model of HOL within ZFC. Types become sets, functions become ZFC functions (sets of ordered pairs), and the indefinite descriptor becomes a choice function on the HOL universe.

2. PART I: CONVERSION FROM OPENTHEORY TO hol.mm

In this part, we present a transformation from OpenTheory article file format to Metamath hol.mm format.

2.1 OpenTheory

Although the original goal of this research was the translation of HOL Light proofs to Metamath, the source of the translation was changed to OpenTheory because it already provides bidirectional translation to HOL Light. Furthermore, the OpenTheory file format is much simpler than the HOL Light file format, requiring only minimal work to read the derivations into an article reader. But more importantly the derivations are all laid out already without needing to be generated first, since HOL Light “proofs” are not actually derivations but instructions for searching for derivations. By targeting OpenTheory we are able to skip the step of deriving the proof to begin with and start from a baseline of a complete proof in HOL-compatible format.

An OpenTheory article file works as a stack machine, and the file format is very simple—a sequence of commands that manipulate the stack, separated by newlines [Hur11]. An article reader maintains a stack, a dictionary for backreferencing previous computations and theorems, and a list \( \Gamma \) of assumptions and an export list \( \Delta \) of theorems. The result of the computation after all instructions in the file are executed is a “theory” \( \Gamma \vdash \Delta \) that states that the theorems in \( \Delta \) are derivable from the axioms in \( \Gamma \). Figure 1 shows the inference rules supported by the logical kernel. The base syntax involves two types of variables: \( t,u,f,g,x,y \) denote term metavariables (and \( \phi,\psi \) denote term metavariables of type \( \text{bool} \)), while \( \alpha,\beta \) are type metavariables. By “metavariable” we mean that an application of any of these axioms will not involve a literal \( t \) but will have \( t \) replaced by some term, which is itself constructed by application of these rules (by contrast to another notion of “metavariable” used in Metamath presentations; see section 2.2). The built-in constants are the type constant \( \text{bool} \), the function type operator \( \alpha \to \beta \), and the equality operator \( = : \alpha \to (\alpha \to \text{bool}) \) (the term \( (= x) \) \( y \) is presented as \( x = y \) for clarity). Each term variable \( v \) is constructed from a name (a string) and a type, and two variables are considered equal only if the name and type are equal.

The structure of a derivation is as a list of theorems of the form \( \Gamma \vdash \phi \), where \( \phi \) is a term of type \( \text{bool} \), and \( \Gamma \) is a set of terms of type \( \text{bool} \). The set unions and differences in \text{deductAntisym} treat terms that are \( \alpha \)-equivalent as equal, where two terms are considered to be \( \alpha \)-equivalent if there is a consistent mapping of bound variables that transforms one term into the other. More precisely:

\textbf{Definition 2.1.} Given a map \( \sigma \) of variables to variables, Terms \( t,u \) are said to be \( \alpha \)-equivalent with respect to \( \sigma \) if one of the following conditions is met:

- \( t = u = c \) for some constant term \( c \)
- \( t = u = v \) for some variable term \( v \) not in the domain of \( \sigma \)
- \( t = v \) and \( u = \sigma(v) \) for some variable term \( v \) in the domain of \( \sigma \)
A does not have v to the definition of a proper substitution, used in rule subst variable of the same type as v replacing any mapping for |Definition 2.2. Given a map \( t, u \) Two terms \( \sigma \) also in the domain of an empty map.

---

\( \text{Fig. 1. The OpenTheory logical kernel.} \)

| Rule | Description |
|------|-------------|
| refl | \( \frac{t : \alpha}{\Gamma \vdash t = t} \) |
| assume | \( \frac{\phi : \text{bool}}{\Gamma \vdash \{\phi\} \vdash \phi} \) |
| subst \( \sigma \) | \( \frac{\Gamma \vdash \phi \quad \{\sigma\} \vdash \phi[\sigma]}{\Gamma \vdash \phi[\sigma]} \) |
| defineConst \( c \) | \( \frac{t : \alpha \quad c : \alpha \quad \Gamma \vdash c = t}{\vdash \phi r = (\text{rep } r) = r} \) |

---

---

- \( t = f x \) and \( u = g y \), and \( f, g \) and \( x, y \) are \( \alpha \)-equivalent with respect to \( \sigma \)
- \( t = (\lambda w. x) \) and \( u = (\lambda v. y) \), and \( x, y \) are \( \alpha \)-equivalent with respect to \( \sigma[w \mapsto v] \), where \( \sigma[w \mapsto v] \) represents the map \( \sigma \) with \( w \mapsto v \) added to the map, replacing any mapping for \( w \) if it exists.

Two terms \( t, u \) are said to be \( \alpha \text{-equivalent} \) if they are \( \alpha \)-equivalent with respect to an empty map.

The term \( \phi' \) in eqMp is required to be \( \alpha \)-equivalent to \( \phi \). This definition is related to the definition of a proper substitution, used in rule subst.

**Definition 2.2.** Given a map \( \sigma \) of term variables to terms and a term \( t \), the proper substitution \( t[\sigma] \) is defined by structural induction as follows:

- For a constant term \( t = c \) or a variable term \( t = v \) not in the domain of \( \sigma \), \( t[\sigma] = t \).
- For a variable term \( t = v \) in the domain of \( \sigma \), \( t[\sigma] = \sigma(v) \).
- If \( t = f x \), then \( t[\sigma] = f[\sigma] x[\sigma] \).
- If \( t = (\lambda v. x) \) and there is some variable present as a subterm of \( t \) that is also in the domain of \( \sigma \), then \( t[\sigma] = (\lambda v. x)[v \mapsto w][\sigma] \), where \( w \) is a new dummy variable of the same type as \( v \), otherwise \( t[\sigma] = (\lambda v. x)[\sigma] \).

---

1The variable \( v \) is assumed to be of type \( \alpha \) in these rules.
2The terms \( \phi, \phi' \) are required to be \( \alpha \)-equivalent.
3The variable \( v \) must not be free in \( \Gamma \). Equivalently, \( \Gamma \) must be \( \alpha \)-equivalent to a term \( \Gamma' \) that does not have \( v \) as a subterm.
4The list of variables \( vs \) must match the set of free variables in \( \phi \); this defines a type operator \( A[vs] \) and constants \( abs : \alpha \to A[vs], \text{rep} : A[vs] \to \alpha \).
For a set $\Gamma$ of terms, $\Gamma[\sigma] = \{t[\sigma] : t \in \Gamma\}$. We use the same notation $t[\sigma]$ for a substitution of type variables to types, but in this case the substitution is direct (distributes through all term and type construction operators).

The OpenTheory stack machine also contains a command $\text{axiom}$ that accepts a list of terms $\Gamma$ and a term $\phi$ and produces the theorem $\Gamma \vdash \phi$, while also adding $\Gamma \vdash \phi$ to the list of assumptions, and a command $\text{thm}$ that takes as input $\Gamma$ and $\phi$ and also a theorem $\Gamma' \vdash \phi'$ where $\Gamma'$ and $\phi'$ are $\alpha$-equivalent to $\Gamma$ and $\phi$, and adds $\Gamma \vdash \phi$ to the list of exported theorems. These rules are sufficient to completely describe the OpenTheory logical kernel.

In addition to the above axioms, there are three axioms that are appended in order to develop the full HOL foundations:

- **extensionality** $\vdash \forall t. (\lambda x. t x) = t$
- **choice** $\vdash \forall p, x. p x \Rightarrow p (\epsilon p)$
- **infinity** $\vdash \exists f^{\text{ind} \rightarrow \text{ind}}. (\text{injective } f \land \neg \text{surjective } f)$

However, these require more advanced definitions such as $\forall$ and $\Rightarrow$, and their translations are straightforward, so can be safely ignored for Part I. These definitions also implicitly introduce the type constant $\text{ind}$ (the intended model is as any infinite set) and the “indefinite descriptor” $\epsilon : (\alpha \rightarrow \text{bool}) \rightarrow \alpha$, which is the HOL equivalent of a global choice function. We will consider these constructs in more detail in Part II.

### 2.2 hol.mm

The primary features of the OpenTheory system (see section 2.1) that are not available in Metamath-based axiomatizations are proper substitution and $\alpha$-equivalence, the “not free in” predicate, and set manipulation of the contexts (the $\Gamma$ in $\Gamma \vdash \phi$).

To address the last problem, we introduce a new term constructor, the “context conjunction”, which takes as input terms $\phi$ and $\psi$ of type $\text{bool}$ and produces a term $(\phi, \psi)$, also of type $\text{bool}$. When $\land$ is defined, it becomes possible to prove that $(\phi \land \psi) = (\phi, \psi)$, but before this it is necessary to introduce this as part of the axiomatization for the “bootstrapping” phase. After taking $\top$ as axiomatic, it becomes possible to represent $\Gamma \vdash \phi$ as simply $\psi \vdash \phi$ by using the context conjunction to put all the terms in $\Gamma$ into one term, and use $\psi = \top$ if $\Gamma = \emptyset$.

Figure 2 shows the axiomatization that is used in hol.mm. A comparison with the axioms of OpenTheory (figure 1) shows several differences. The most obvious difference is the increase in the number of axioms, from 14 to 23, but this comparison is deceptive, because this increase must be weighed against the more complicated explanation of $\alpha$-equivalence and proper substitution that must be described in order to fully explain when the axioms are applicable. In the hol.mm axiomatization, the only “asterisks” are the distinct variable provisos that come with leq, ax-17, and distrl.

There are three kinds of variables in these axioms: “type”, “var”, and “term” variables. Term variables are represented by capital letters, type variables are represented with greek letters, and “var” variables (referred to henceforth as vars) are represented by lowercase letters. The reason for the division of OpenTheory term variables into two different types has to do with the way which Metamath
handles substitutions. Metamath is designed to have “pluggable” axioms, with the only built-in axiom being the direct substitution of a (meta)variable for a term. This substitution process happens for every application of every theorem, and the only restriction on substitutions is that the substitution must be for a term, and the distinct variable provisos must be honored.

Unlike OpenTheory, vars in hol.mm do not come with built-in type information, so a variable term takes a var as well as a type; this is represented as a superscript in figure 2 (i.e. $x^\alpha$). Note that even vars like $x$ are metavariables, but they can only be substituted for other vars (because the only expressions of type “var” are other vars). The capital term variables, on the other hand, can be substituted for other terms, like $x^\alpha$ or $(\lambda x^\alpha.A)$. In Metamath, it is easier to work with term variables than vars because they admit direct substitution, but only vars can be bound in constructs like lambda abstractions, so both variable types are necessary in the

---

4The variable $x$ is required to not be present in $R$.
5The variable $x$ is required to not be present in $A$.
6The variable $y$ is required to be distinct from $x$ and not present in $B$.
Metamath interpretation. For simplicity of presentation, however, we will stick to using vars only during a derivation and keep the metavariables to just the axioms themselves.

Most of the axioms have direct equivalents, but a few deserve extra explanation. The axioms \texttt{wtru}, \texttt{wct}, \texttt{id}, \texttt{syl}, \texttt{jca}, \texttt{simpl}, \texttt{simpf}, \texttt{trud} are necessary in order to handle context manipulation: doing the set unions in OpenTheory axioms like \texttt{eqMp} (this is discussed in more detail in section 2.4). The axioms \texttt{hb1l}, \texttt{distc}, \texttt{distrl}, \texttt{ax-17} are used to define the properties of the “not free in” predicate, which is used in \texttt{inst}.

**Definition 2.3.** Let \( \text{NF}(x, A) \) denote the statement that \( \top \vdash (\lambda x^\alpha. A) y^\alpha = A \) is derivable (with explicit metavariables \( y, \alpha \)). We can read this as “\( x \) is not free in \( A \)”.

Then \texttt{ax-17} asserts that \( \text{NF}(x, A) \) whenever \( A \) is a term that is distinct from \( A \), and \texttt{hb1l} asserts that \( \text{NF}(x, (\lambda x^\alpha. A)) \), while \texttt{distc} and \texttt{distrl} can be used to show that \( \text{NF}(x, A) \) and \( \text{NF}(x, B) \) imply \( \text{NF}(x, A B) \) and \( \text{NF}(x, (\lambda y^\alpha. A)) \) (when \( x \) and \( y \) are distinct). Thus this predicate allows one to represent the structural property “\( x \) is not free in \( A \)” faithfully within the logic. (This same trick is used in \texttt{set.mm} to represent the not-free predicate, although there it is expressed more naturally as \( (\varphi \rightarrow \forall x \varphi) \). The name \texttt{ax-17} is borrowed from \texttt{set.mm}, where an axiom by the same name asserts \( (\varphi \rightarrow \forall x \varphi) \) when \( x \) is distinct from \( \varphi \).)

The three extra logical axioms in HOL are translated to the following three axioms:

\begin{align*}
\text{eta} & \quad \top \vdash \forall f^{\alpha \rightarrow \beta}. (\lambda x^\alpha. f^{\alpha \rightarrow \beta} x^\alpha) = f^{\alpha \rightarrow \beta} \\
\text{ac} & \quad \top \vdash \forall p^{\alpha \rightarrow \text{bool}}, \forall x^\alpha. (p^{\alpha \rightarrow \text{bool}} x^\alpha \Rightarrow p^{\alpha \rightarrow \text{bool}} (\epsilon p^{\alpha \rightarrow \text{bool}})) \\
\text{inf} & \quad \top \vdash \exists f^{\text{ind} \rightarrow \text{ind}}. (\text{injective } f^{\text{ind} \rightarrow \text{ind}} \land \neg \text{surjective } f^{\text{ind} \rightarrow \text{ind}})
\end{align*}

These axioms are exactly what you would expect from the earlier listing on page 4.

Also, the following simple theorems of the system will be useful:

\begin{align*}
\top \vdash A & \quad \text{ali} \\
R \vdash A = B & \quad \text{eqcomi} \\
R \vdash A = B & \quad \text{eqtri} \\
R \vdash A = B & \quad \text{cteq} \\
A : \text{bool} & \quad C : \text{bool} \quad R \vdash A = B \quad R \vdash C = D & \quad \text{hbct} \\
R \vdash (A, C) = (B, D) & \quad \text{cbv}
\end{align*}

In the last theorem, \( y \) must not be present in \( A \) and \( x \) must not be present in \( B \). This theorem is curious because although it is true using only OpenTheory core theorems (trivially, since \( (\lambda x^\alpha. A) \) and \( (\lambda y^\alpha. B) \) are \( \alpha \)-equivalent), it requires \texttt{eta} for its \texttt{hol.mm} proof.
2.3 Language embedding

Our first step in defining the conversion from OpenTheory to hol.mm is to define the embedding of formulas in one language into formulas in the other. We denote this function as $F$.

**Definition 2.4.** The map $F$ operates on statements, terms, and types of the OpenTheory language, and outputs statements, terms, and types of the hol.mm language.

- For a constant term or type $c$, $F(c) = c$, where a constant named $c$ is defined in the file if it is not already present.
- For a variable term $v$ of type $\alpha$, $F(v) = v^{F(\alpha)}$ and $F(\lambda v. x) = (\lambda v^{F(\alpha)}. F(x))$, where a var named $v$ is defined in the file if it is not already present.
- For a type variable $\alpha$, $F(\alpha) = \alpha$, where a type variable named $\alpha$ is defined in the file if it is not already present.
- $F(f x) = F(f) F(x)$
- $F(t : \alpha) = F(t) : F(\alpha)$
- $F(\alpha \rightarrow \beta) = F(\alpha) \rightarrow F(\beta)$
- If $A[\alpha_1, \ldots, \alpha_n]$ is a type operator of arity $n$, then $F(A[\alpha_1, \ldots, \alpha_n]) = A[F(\alpha_1), \ldots, F(\alpha_n)]$, and $A[\beta_1, \ldots, \beta_n]$, with literal type variables $\beta_1, \ldots, \beta_n$, is added as a syntax constructor if it is not already present (and $\beta_1, \ldots, \beta_n$ are added as type variables if not present).
- $F(\{\phi_1, \phi_2, \ldots, \phi_n\} \vdash \psi) = (F(\phi_1), F(\phi_2), \ldots, F(\phi_n)) \vdash F(\psi)$, unless $n = 0$ in which case $F(\vdash \psi) = \top \vdash F(\psi)$.

**Remark 2.1.** Since sets are unordered but the context conjunction is, $F(\Gamma \vdash \phi)$ is not uniquely defined. However, any of the choices of ordering of $\Gamma$ produce equivalent statements, and internally $\Gamma$ is usually stored as a list anyway, so one may as well use this ordering. Alternatively, one can define a total order on terms and insist that the listing be done in increasing order to ensure uniqueness. In any case, the ordering chosen will not be relevant to later developments.

In the course of “evaluating” this function on the statements of an OpenTheory derivation, at various points certain variables and constants will be added to the logical system. This is necessary because all variable and constant names need to be predeclared in a Metamath file, so this ensures that the predefinitions are made and allows an OpenTheory file to use variables that may not have been defined in the hol.mm core (which only defines variables that are used in the axioms themselves, such as $x, y, A, B, \alpha, \beta$). Any other variable names, like $v$, will need to be declared before their use in a theorem).

We assume that all term variable names in an OpenTheory derivation are distinct from type variable names and hol.mm core axiom and theorem labels, and no variable is used multiple times in the same theorem statement with different types, because OpenTheory will consider these distinct while hol.mm will consider them the same (i.e. $\text{NF}(x, (\lambda x^\alpha. x^\beta))$ even though OpenTheory would consider $x^\beta$ as free in that expression). This can be ensured with suitable pre-processing of the OpenTheory article file.
Now we are finally capable of stating the main goal of this part, although the proof will be postponed to the next section.

**Theorem 2.1.** If the statement $\Gamma \vdash \phi$ is derivable in OpenTheory, then $F(\Gamma \vdash \phi)$ is derivable in a conservative extension of hol.mm.

The reason for the “conservative extension” caveat is because in addition to the variables and constants being added to the system by $F$, our transformation will also need to add definitions coming from `defineConst` and `defineTypeOp`, and Metamath does not support a special definition construct. Instead, definitions and axioms are treated on equal footing, and an external tool can be used to show that the axioms that claim to be definitions are actually conservative.

### 2.4 Proving the embedded OpenTheory axioms

Our proof of theorem 2.1 will proceed by defining an explicit map from OpenTheory derivations to `hol.mm` derivations. First, we show that proper substitution, simplification, and $\alpha$-equivalence are derivable.

**Lemma 2.1.** If $A$ is a nested context conjunction containing $B$ as a conjunct, then $A \vdash B$ is provable.

Proof. By induction on the length of $A$. If $A = B$, then id proves $A \vdash B$. If $A = (A_1, A_2)$ and $B$ is a conjunct of $A_1$, then `simpl` proves $A \vdash A_1$ and by induction $A_1 \vdash B$ is provable, so `syl` proves $A \vdash B$. The case of $B$ a conjunct of $A_2$ is similar (using `simp` instead).

**Lemma 2.2.** If $A$ does not contain $x$ and $C = B[x/A]$ is the result of the proper substitution of $A$ for $x^\alpha$ in $B$ (where proper substitution of `hol.mm` terms is defined similarly to OpenTheory proper substitution), then $x^\alpha \vdash A \vdash B = C$ is provable, and $\text{NF}(x, C)$.

Proof. By induction on the length of $B$.

— If $B$ does not contain $x^\alpha$, then $C$ is identical to $B$ and so `refl`, `a1i` proves $x^\alpha \vdash A \vdash B = B$ and `ax-17` proves $\text{NF}(x, B)$.

— If $B = x^\alpha$, then $C = A$ so `id` proves $x^\alpha \vdash A \vdash x^\alpha = A$ and `ax-17` proves $\text{NF}(x, A)$.

— If $B = B_1 \ B_2$, then $C = B_1[x/A] \ B_2[x/A]$, so the induction hypothesis gives proofs of $x^\alpha \vdash A \vdash B_1 = B_1[x/A]$ and $x^\alpha \vdash A \vdash B_2 = B_2[x/A]$, and `ceq` proves $x^\alpha \vdash A \vdash B = C$ and `distrc`, `ceq` prove $\text{NF}(x, C)$ from $\text{NF}(x, B_1[x/A])$ and $\text{NF}(x, B_2[x/A])$. (The same is true when $B = (B_1, B_2)$, with `cteq` in place of `ceq` and `hbet` for the proof of $\text{NF}(x, C)$.)

— If $B = (\lambda x^\beta. \ B_1)$, then $\text{NF}(x, B_1)$, so $C$ is identical to $B$ and so again `refl`, `a1i` proves $x^\alpha \vdash A \vdash B = B$ and $\text{NF}(x, B)$ is given.

— If $B = (\lambda y^\beta. \ B_1)$ where $x, y$ are distinct, then $C = (\lambda y^\beta. \ B_1[x/A])$ and `distrl`, `leq`, `eqtri` proves $\text{NF}(x, C)$. If $A$ does not contain $y$, then `leq` proves the goal. If $A$ does contain $y$, then `leq` is not directly applicable, and the proper substitution for $C = (\lambda z^\beta. \ C_1)$ includes a dummy variable $z$. In this case, use the induction hypothesis to prove $y^\beta = z^\beta \vdash B_1 = B_1[y/z]$, and then apply `cbv`, `a1i` to get $x^\alpha \vdash A \vdash B = (\lambda z^\beta. \ B_1[y/z])$. Using the induction hypothesis once more to prove...
\[ x^\alpha = A \vdash B_1[y/z] = C_1, \text{ leq gives } x^\alpha = A \vdash (\lambda z^\beta. B_1[y/z]) = C \text{ and eqtri proves the goal theorem.} \qedhere \]

**Lemma 2.3.** If \( A \) is \( \alpha \)-equivalent to \( B \) (where \( \alpha \)-equivalence of hol.mm terms is defined similarly to OpenTheory \( \alpha \)-equivalence), then \( \top \vdash A = B \) is provable.

**Proof.** By induction on the length of \( A \).

- If \( A \) is a variable or constant, then \( A = B \) and \( \text{refl} \) proves the goal.
- If \( A = A_1 A_2 \), then \( B = B_1 B_2 \) and \( \text{ceq} \) proves the goal.
- If \( A = (A_1, A_2) \), then \( B = (B_1, B_2) \) and \( \text{cteq} \) proves the goal.
- If \( A = (\lambda x^\alpha. A_1) \) and \( B = (\lambda x^\alpha. B_1) \), then \( \text{leq} \) proves the goal.
- If \( A = (\lambda x^\alpha. A_1) \) and \( B = (\lambda y^\alpha. B_1) \), then let \( A'_1 = A_1[x/y] \) and \( A' = (\lambda y^\alpha. A'_1) \). Then \( A \) is \( \alpha \)-equivalent to \( A' \) and \( B \), so by the third clause \( \top \vdash A' = B \) is provable, and by lemma 2.2 \( x^\alpha = y^\alpha \vdash A_1 = A'_1 \) is provable, so \( \text{cbv} \) gives \( \top \vdash A = A' \) and \( \text{eqtri} \) proves the goal. \( \Box \)

**Lemma 2.4.** If \( A \) is \( \alpha \)-equivalent to \( B \) and \( C \) is \( \alpha \)-equivalent to \( D \), then \( A \vdash C \) implies \( B \vdash D \).

**Proof.** Lemma 2.3 applied twice gives us \( \top \vdash A = B \) and \( \top \vdash C = D \), and \( \text{id, ali, eqmp, eqcomi} \) turn these into \( B \vdash A \) and \( C \vdash D \), and then \( \text{syl} \) gives \( B \vdash D \) as desired. \( \Box \)

In order to prove Theorem 2.1, we cast it as a special case of a more general theorem, using an invariant property which we’ll call **reduction**.

**Definition 2.5.** Given an OpenTheory term \( \phi \) and a hol.mm term \( A \), we say that \( A \) **reduces to** \( \phi \) if there is a \( \phi' \) \( \alpha \)-equivalent to \( \phi \) with \( \mathcal{F}(\phi') = A \), and given an OpenTheory statement \( \Gamma \vdash \phi \) and a hol.mm statement \( A \vdash B \), we say that \( A \vdash B \) **reduces to** \( \Gamma \vdash \phi \) if there is a type variable \( \sigma \) such that \( B \) reduces to \( \phi[\sigma] \) and for every \( \psi \in \Gamma \) either \( A \) reduces to \( \psi[\sigma] \) or \( A = (A_1, A_2) \) and at least one of \( A_1, A_2 \) reduces to \( \psi[\sigma] \).

**Remark 2.2.** Note that \( \mathcal{F}(\Gamma \vdash \phi) \) always reduces to \( \Gamma \vdash \phi \).

Intuitively, the notion of reduction from \( A \vdash B \) to \( \Gamma \vdash \phi \) means that \( \Gamma \) is equivalent to a subset of the conjunction of terms in \( A \), and \( B \) and \( \phi \) are equivalent. Thus if \( \Gamma \vdash \phi \) is provable, then \( A \vdash B \) has only added irrelevant antecedents, so it ought to be provable as well. This forms our main invariant across the derivation.

**Theorem 2.2.** If \( \Gamma \vdash \phi \) is derivable in OpenTheory and \( A \vdash B \) reduces to \( \Gamma \vdash \phi \), then \( A \vdash B \) is derivable in a conservative extension of hol.mm, and if \( t : \alpha \) is derivable in OpenTheory, then \( \mathcal{F}(t : \alpha) \) is derivable in a conservative extension of hol.mm.

**Proof.** The proof is by induction on the length of the proof of the OpenTheory statement. We break the proof into cases based on the last inference rule in the derivation tree.
2.4.1 Direct conversions. Many of the axioms are converted directly into equivalent axioms. Specifically:

- \( \text{varTerm} \rightarrow \text{wv} \)
- \( \text{absTerm} \rightarrow \text{wl} \)
- \( \text{appTerm} \rightarrow \text{wc} \)
- \( \text{refl} \rightarrow \text{refl} \)
- \( \text{eqMp} \rightarrow \text{eqmp} \)
- \( \text{appThm} \rightarrow \text{ceq} \)
- \( \text{deductAntisym} \rightarrow \text{ded} \)
- \( \text{extensionality} \rightarrow \text{eta} \)
- \( \text{choice} \rightarrow \text{ac} \)
- \( \text{infinity} \rightarrow \text{inf} \)

Given derivations of all the hypotheses to one of these inferences, apply the transformed step to the transformed hypotheses (and \( \alpha \)-to the result if \( A \neq \top \)) to get an \( \alpha \)-equivalent statement, and the lemma 2.4 finishes the job.

2.4.2 assume. This axiom is an application of lemma 2.1 to prove \( A \vdash B' \) where \( B' \) is the conjunct of \( A \) that is \( \alpha \)-equivalent to \( B \), followed by lemma 2.4.

2.4.3 absThm. This axiom is almost a direct application of \( \text{leq} \), but the requirement is only that \( \Gamma \) not have \( v \) free in it, not that \( v \) be completely disjoint from \( \Gamma \). However, if \( v \) is not free in \( \Gamma \), then there is a \( \Gamma' \) that is disjoint from \( v \) and \( \alpha \)-equivalent to \( \Gamma \), so \( \text{leq} \) proves \( F(\Gamma') \vdash (\lambda x^\alpha . A) = (\lambda x^\alpha . B) \) from \( F(\Gamma') \vdash A = B \) and lemma 2.4 applied before and after turn the \( \Gamma' \) into \( \Gamma \) in this inference.

2.4.4 subst. There are two kinds of substitution performed by \( \text{subst} \) — type variable substitution and term variable substitution. If a type variable substitution is performed, so that \( \Gamma = \Gamma' [\sigma] \) and \( \phi = \phi' [\sigma] \), then \( \Gamma' \vdash \phi' \) is also reducible to \( A \vdash B \), so \( A \vdash B \) is provable.

If \( \sigma = [x_1 \rightarrow A_1, \ldots, x_n \rightarrow A_n] \) is a term variable substitution, then by writing this as a composition of \( [x_1 \rightarrow y_1, \ldots, x_n \rightarrow y_n] \) with \( [y_1 \rightarrow A_1, \ldots, y_n \rightarrow A_n] \) where \( y_i \) are dummy variables, we can reduce this to the case when no \( A_i \) contains any \( x_j \). Then we can rewrite it again as the composition of \( \sigma_1 = [x_1 \rightarrow A_1], \ldots, \sigma_n = [x_n \rightarrow A_n] \), so that we can reduce to the case of a single variable substitution. And this case is handled by axiom \( \text{inst} \), with the hypotheses filled by lemma 2.2.

2.4.5 betaConv. This axiom is an application of \( \text{beta} \) followed by the same substitution process described in section 2.4.4 (and \( \text{a1i} \), lemma 2.4).

2.4.6 Definitions. Lastly, we have the two definitional axioms, \text{defineConst} and \text{defineTypeOp}. In this case, we simply introduce all the output statements as axioms. We can do a little better, though; by introducing the axioms

\[
\frac{B : \alpha \quad R \vdash A = B}{A : \alpha} \quad \text{eqtypri}
\]

\[
\frac{\top \vdash F B \quad \text{TD}_{\beta, A . R}(F, B)}{A : \alpha \rightarrow \beta \quad R : \beta \rightarrow \alpha \quad \top \vdash (A; (R x^\beta) = x^\beta, F y^\alpha = (R (A y^\alpha) = y^\alpha))} \quad \text{typedef}
\]

Journal of Formalized Reasoning Vol. ?, No. ?, Month Year.
we can set it up so that the only axiom needed for \texttt{defineConst} is a single axiom $\top \vdash c = t$ for the constant’s definition, and the only axiom needed to define a new type is $\text{TD}_{\beta, A, R}(F, B)$ (which is a new kind of statement designed solely for input to \texttt{typedef}). However the checking that the constant does not appear in the definition, the typedef’s type constant $\beta$ lists all free type variables in use, etc. must still be checked outside the system. Nonetheless, as long as the original OpenTheory derivation followed these consistency rules, the transformed definition will also be conservative, for the same reasons, so it does not interfere with this proof.

\textit{Proof of Theorem 2.1.} Follows immediately from Remark 2.2 and Theorem 2.2.

3. PART II: CONVERSION FROM \texttt{hol.mm} TO \texttt{set.mm}

In this part, we have the remaining job of transforming our Metamath representation of a HOL axiomatic system into ZFC. Even though the foundations are changing, since the basic functions of substitution and the like are the same on both the start and endpoint of the transformation, we can focus on the mathematical content of the sentences without worrying as much about the exact representation of the formula. We begin by describing the model of HOL that we will build in ZFC:

\textbf{Definition 3.1.} A \textit{type} $\alpha$ is a pair $\langle \iota_{\alpha}, b_{\alpha} \rangle$ of a \textit{witness} and a \textit{base set} such that $b_{\alpha} \in V_{\omega}^{2}$ and $\iota_{\alpha} \in b_{\alpha}$. Let Ch($\epsilon$) denote that either $\epsilon$ is a choice function on $V_{\omega}^{2}$ or there is no such function, and define a map $\mathcal{S}_\epsilon$ from types, terms and statements of \texttt{hol.mm} to sets and wffs of \texttt{set.mm}:

- For the two constant types, we take $\mathcal{S}_\epsilon(\text{bool}) = \langle 1, 2 \rangle := \langle 1, \{0, 1\} \rangle$ and $\mathcal{S}_\epsilon(\text{ind}) = \langle 0, \omega \rangle$, and define $\mathcal{S}_\epsilon(\top) = 1$.
- For type variables, $\mathcal{S}_\epsilon(\text{type} \alpha) \leftrightarrow \alpha$ is a type; this hypothesis is implicit in all axioms that involve type variables.
- For convenience, define the wff predicate $\text{toWff}(A) \iff A = 1$, and the function $\text{toBool}(\varphi) = \text{if}(\varphi, 1, 0)$.
- A variable is mapped to $\mathcal{S}_\epsilon(x^\alpha) = \text{if}(x \in b_{\alpha}, x, \iota_{\alpha})$.
- For the function type, we take $\mathcal{S}_\epsilon(\alpha \to \beta) = \langle (x \in b_{\alpha} \mapsto \iota_{\beta}), b_{\beta} \rangle$, the set of all ZFC functions from $\alpha$ to $\beta$, with a constant function as witness.
- Function application is represented by function application: $\mathcal{S}_\epsilon(F \ A) = (F' A)$.
- Lambda abstraction is represented by the mapping operator $\mathcal{S}_\epsilon(\lambda x^\alpha. A) = (x \in b_{\alpha} \mapsto A)$.
- Context conjunction is represented by conjunction: $\mathcal{S}_\epsilon((A, B)) = \text{toBool}(\text{toWff}(A) \land \text{toWff}(B))$.
  - For a term, $\mathcal{S}_\epsilon(A : \alpha) \mapsto A \in b_{\alpha}$.
  - For a theorem, $\mathcal{S}_\epsilon(A \vdash B) \mapsto (\text{Ch}(\epsilon) \land \text{toWff}(A)) \rightarrow \text{toWff}(B)$.
- The equality operator $=_{\alpha}$ is mapped, depending on its type, to $\mathcal{S}_\epsilon(=_{\alpha}) = (x \in b_{\alpha} \mapsto (y \in b_{\alpha} \mapsto \text{toBool}(x = y)))$
  - The indefinite descriptor $\epsilon_{\alpha}$ itself is mapped, depending on its type, to $\mathcal{S}_\epsilon(\epsilon_{\alpha}) = (f \in b_{\alpha}^{0, 1} \mapsto \text{if}(\forall x \in b_{\alpha}. f(x) = 0, \iota_{\alpha}, \epsilon_{\alpha}(\{x \in b_{\alpha} : f(x) = 1\}))$.

\textit{Journal of Formalized Reasoning Vol. ?, No. ?, Month Year.}
— A defined type \( S_\epsilon(TD_{\beta,A,R}(F,B)) \) asserts that \( \beta = \langle B, \{x \in b_\alpha : \text{toWff}(F(x))\} \rangle \), \( A = (x \in b_\alpha \mapsto \text{if}(x \in b_\beta, x, B)) \), and \( R = (x \in b_\beta \mapsto x) \).

Most of these definitions are exactly what you would expect—functions are functions, and types and terms map to sets and their elements. The unusual part of the definition deals with the indefinite descriptor \( \epsilon \). HOL is based on a version of the Axiom of Choice that is stronger than the usual one in ZFC. Instead of asserting that for any set there exists a choice function on that set, it asserts that a specific function is a choice function on the universe. If the HOL universe were a proper class, this would be problematic, but luckily it can be entirely contained within \( V_{\omega2} \), which is a set in ZFC, and thus the ZFC axiom of choice gives us a single choice function \( \epsilon \) on all of \( V_{\omega2} \). We pass this in as a parameter to \( S \), so that we can give meaning to the various pieces of the formula that use \( \epsilon \), and theorems assert the choice behavior of \( \epsilon \), so that it can be used in derivations. The reason for the “or if there is no such function” proviso is to allow the proof of \( \vdash S_\epsilon(T \vdash A) \implies \vdash \text{toWff}(A) \) to avoid choice, so that the ZFC choice axiom never gets invoked unless \( \text{ac} \) does. Since we also require a term var to have unconditional closure, we are forced to add witnesses so that you don’t need to invoke choice by using \( \epsilon \) to select elements.

3.1 Proving the hol.mm axioms

**Theorem 3.1.** If \( A \vdash B \) is derivable in hol.mm, then \( \vdash S_\epsilon(A \vdash B) \) is derivable in set.mm. In particular, if \( A \vdash B \) does not contain \( \epsilon \), then \( \vdash S(A) \rightarrow S(B) \) is derivable (where the \( \epsilon \) has been dropped from the notation to indicate that the action of \( S \) does not depend on \( \epsilon \)).

**Proof.** Here we merely need to verify that each of the axioms is preserved under wrapping by \( S_\epsilon \). Note that \( S_\epsilon \) can actually be defined in set.mm, so that each of the axioms, in \( S_\epsilon \)-wrapped form, can be proven as theorems within set.mm, and then the transformation will be one-to-one in terms of proof length. Also, keep in mind that there are implicit hypotheses that \( \alpha, \beta, \) etc. are types; these become explicit hypotheses during this translation. Abbreviated proofs for each axiom are presented, using theorem labels for existing set.mm proofs when necessary.

— type bool: \( 1 \in 2 \in V_{\omega2} \)
— type ind: \( 0 \in \omega \in V_{\omega2} \)
— type(\( \alpha \rightarrow \beta \)): If \( \text{rank}(b_\alpha) = m < \omega2 \) and \( \text{rank}(b_\beta) = n < \omega2 \), then \( \text{rank}(b_\alpha^{b_\beta}) \leq \max(m,n) + 3 < \omega2 \)
— wtru: \( 1 \in 2 \)
— wct: toBool of anything is in 2
— wc: If \( f : \alpha \rightarrow \beta \) and \( x \in \alpha \), then \( f(x) \in \beta \)
— wv: If \( x \in b_\alpha \), then \( (x \in b_\alpha \mapsto x) \in b_\alpha \), otherwise \( \text{iota}_\alpha \in b_\alpha \)
— wl: If for all \( x, T \in b_\beta \), then \( (x \in \alpha \mapsto T) : b_\alpha \rightarrow b_\beta \)
— id: theorem simpr
— syl: theorem syldan
— jca: theorem jca
— weq: use the definition and two applications of wl
— simpl, simpr: theorem simpl, simpr
— trud: theorem a1i, tru
— refl: theorem eqidd
— eqmp: theorem mpbid after showing toWff(A = B) ↔ (toWff(A) ↔ toWff(B)) by case analysis
— ded: theorem impbid, expr
— ceq: theorem fveq12d
— leq: theorem mpteq2dv
— hbl1: theorem hbmpt1
— distr, distr: mapping theorems
— ax-17: theorem fvmpt, eqidd
— beta: theorem fvmpt2
— inst: theorem vtoclf
— eta: theorem dfnn5v
— inf: short proof using (x ∈ ω ↦→ x + 1)
— ac: By definition, εα(p) = if(∀x ∈ bα, p(x) = 0, ε(\{x ∈ bα : p(x) = 1\})), but since p(x) = 1, the if-condition is false, so εα(p) = ε(\{x ∈ bα : p(x) = 1\}). Assuming ax - ac, ε is a choice function on Vω2, and \{x ∈ bα : p(x) = 1\} is a nonempty subset of bα ∈ Vω2, so ε(\{x ∈ bα : p(x) = 1\}) ∈ \{x ∈ bα : p(x) = 1\} and thus p(εα(p)) = 1.

For the final statement, observe that no S transformation depends on ε except Sε(εα), so if ε is not in A or in B then it will not be in the right hand side of Ch(ε) → (S(A) → S(B)), and theorem exlimiv turns this into ∃ε. Ch(ε) → (S(A) → S(B)). But ∃ε. Ch(ε) is provable, because if there is a choice function ε on Vω2 then Ch(ε) for that choice of ε and if not then Ch(0) is true. □

4. FUTURE WORK

We stopped at Part II here, but one can argue the existence of a Part III to this translation project, where notations such as the HOL Light natural numbers are mapped via the natural isomorphisms to the set.mm natural numbers, so that a statement like the Prime Number Theorem, which is (as of this writing) proven in HOL Light but not in set.mm, can be said to be proven in its “natural form”, rather than in some model. This process is much less formulaic, however, and requires individual considerations of each mathematical concept in order to identify the proper isomorphisms.

A programmatic implementation exists for some of Part I and some of Part II, but was not able to be completed in time for the submission deadline. Hopefully it will be complete by the time of the revisions, so that data on relative efficiencies and drawbacks of the implementation can be reported.

ACKNOWLEDGMENTS

The author wishes to thank Norman Megill, Bob Solovay, and Raph Levien for their work on investigating Part II in many email discussions, as well as John Harrison.
for creating HOL Light and Joe Leslie-Hurd for creating OpenTheory and providing the source material for figure 1.

References

[Far08] William M Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.

[Har09] John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66, Munich, Germany, 2009. Springer-Verlag.

[Hur11] Joe Hurd. The OpenTheory standard theory library. pages 177–191, 2011.

[Kun11] Ondřej Kunčar. Proving valid quantified boolean formulas in HOL Light. *Lecture Notes in Computer Science*, 6898:184–199, 2011.

[Meg07] Norman D. Megill. *Metamath: A Computer Language for Pure Mathematics*. Lulu Publishing, Morrisville, North Carolina, 2007. HTTP://US.METAMATH.ORG/DOWNLOADS/METAMATH.PDF.