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/HOL, and Isabelle/HOL) 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`, as a roadmap for an eventual implementation.

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 Γ of assumptions and an export list Δ of theorems. The result of the computation after all instructions in the file are executed is a "theory" $\Gamma \triangleright \Delta$ that states that the theorems in Δ are derivable from the axioms in Γ. 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 ϕ, ψ denote term metavariables of type bool), while α, β 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 bool, the function type operator $\alpha \to \beta$, and the equality operator $= : \alpha \to (\alpha \to \mathsf{bool})$ (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.

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

Definition 2.1. Given a map σ of variables to variables, terms t, u are said to be *α-equivalent with respect to σ* 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 σ

— $t = v$ and $u = \sigma(v)$ for some variable term v in the domain of σ

— $t = f\ x$ and $u = g\ y$, and f, g and x, y are α-equivalent with respect to σ

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

Two terms t, u are said to be *α-equivalent* if they are α-equivalent with respect to an empty map.

$$\frac{}{v : \alpha}\ \mathsf{varTerm}\ v^1 \qquad \frac{t : \beta}{(\lambda v.\ t) : \alpha \to \beta}\ \mathsf{absTerm}\ v^1 \qquad \frac{f : \alpha \to \beta \quad x : \alpha}{f\ x : \beta}\ \mathsf{appTerm}$$

$$\frac{t : \alpha}{\vdash t = t}\ \mathsf{refl} \qquad \frac{\phi : \mathsf{bool}}{\{\phi\} \vdash \phi}\ \mathsf{assume} \qquad \frac{\Gamma \vdash \phi' = \psi \quad \Delta \vdash \phi}{\Gamma \cup \Delta \vdash \psi}\ \mathsf{eqMp}^2$$

$$\frac{f : \alpha \to \beta \quad x : \alpha \quad \Gamma \vdash f = g \quad \Delta \vdash x = y}{\Gamma \cup \Delta \vdash f\ x = g\ y}\ \mathsf{appThm} \qquad \frac{\Gamma \vdash t = u}{\Gamma \vdash (\lambda v.\ t) = (\lambda v.\ u)}\ \mathsf{absThm}\ v^3$$

$$\frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{(\Gamma - \{\psi\}) \cup (\Delta - \{\phi\}) \vdash \phi = \psi}\ \mathsf{deductAntisym} \qquad \frac{((\lambda v.\ t)\ u) : \alpha}{\vdash (\lambda v.\ t)\ u = t[v \to u]}\ \mathsf{betaConv}$$

$$\frac{\Gamma \vdash \phi}{\Gamma[\sigma] \vdash \phi[\sigma]}\ \mathsf{subst}\ \sigma \qquad \frac{t : \alpha}{c : \alpha \quad \vdash c = t}\ \mathsf{defineConst}\ c$$

$$\frac{t : \alpha \quad \vdash \phi\ t}{\vdash abs\ (rep\ a) = a \quad \vdash \phi\ r = (rep\ (abs\ r) = r)}\ \mathsf{defineTypeOp}\ A[vs]\ abs\ rep^4$$

Fig. 1. The OpenTheory logical kernel.[5]

The term ϕ' in eqMp is required to be α-equivalent to ϕ. This definition is related to the definition of a proper substitution, used in rule subst.

Definition 2.2. Given a map σ 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 σ, $t[\sigma] = t$.

— For a variable term $t = v$ in the domain of σ, $t[\sigma] = \sigma(v)$.

— If $t = f\ x$, then $t[\sigma] = f[\sigma]\ x[\sigma]$.

— If $t = (\lambda v.\ x)$ and some variable present as a subterm of t (this includes v) is also in the domain of σ, then $t[\sigma] = (\lambda w.\ x[v \mapsto w][\sigma])$, where w is a dummy variable distinct from variables in subterms of t of the same type as v, otherwise $t[\sigma] = (\lambda v.\ x[\sigma])$.

For a set Γ 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).

[1]The variable v is assumed to be of type α in these rules.

[2]The terms ϕ, ϕ' are required to be α-equivalent.

[3]The variable v must not be free in Γ. Equivalently, Γ must be α-equivalent to a set of terms Γ' that do not have v as a subterm.

[4]The list of variables vs must match the set of free variables in ϕ; this defines a type operator $A[vs]$ and constants $abs : \alpha \to A[vs]$, $rep : A[vs] \to \alpha$.

[5]The astute reader will notice that this table is more detailed than figure 2 of [Hur11], upon which it was based. This is due to the inclusion of explicit term formation axioms varTerm, absTerm, appTerm; these were elided in the presentation of [Hur11] but are included in later versions of the specification (see `http://gilith.com/research/opentheory/article.html`).

The OpenTheory stack machine also contains a command axiom that accepts a list of terms Γ and a term ϕ and produces the theorem $\Gamma \vdash \phi$, while also adding $\Gamma \vdash \phi$ to the list of assumptions, and a command thm that takes as input Γ and ϕ and also a theorem $\Gamma' \vdash \phi'$ where Γ' and ϕ' are α-equivalent to Γ and ϕ, 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	$\vdash \exists f^{\mathsf{ind}\to\mathsf{ind}}.\ (\mathsf{injective}\ f \wedge \neg \mathsf{surjective}\ f)$

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 ind (the intended model is as any infinite set) and the "indefinite descriptor" $\epsilon : (\alpha \to \mathsf{bool}) \to \alpha$, 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 α-equivalence, the "not free in" predicate, and set manipulation of the contexts (the Γ in $\Gamma \vdash \phi$). To address the last problem, we introduce a new term constructor, the "context conjunction", which takes as input terms ϕ and ψ of type bool and produces a term (ϕ, ψ), also of type bool. When $\wedge$ is defined, it becomes possible to prove that $(\phi \wedge \psi) = (\phi, \psi)$, 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 Γ into one term, and use $\psi = \top$ if $\Gamma = \emptyset$.

Figure 2 shows the axiomatization that is used in `hol.mm`.[1] 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 α-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 with a term.

[1] A version of `hol.mm` is available for download at `http://us.metamath.org/metamath/hol.mm`.

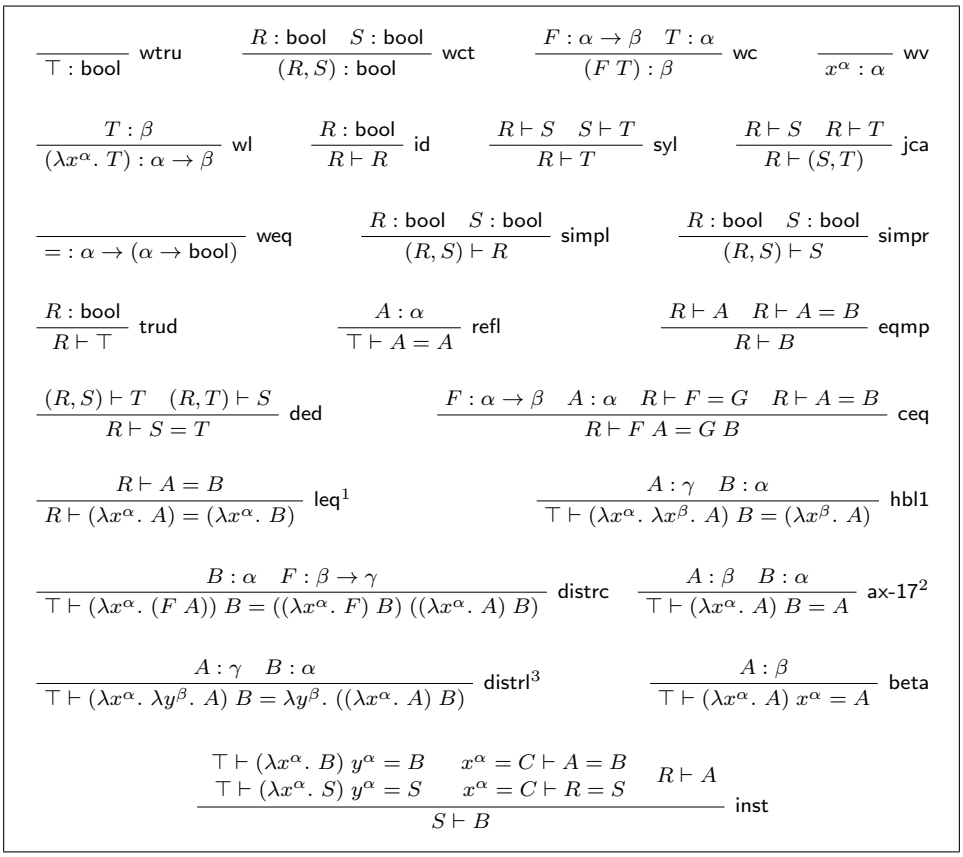

Fig. 2. The `hol.mm` axiomatization.

This substitution process happens for every application of every theorem, and the only restriction on substitutions is that the substitution must be with 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 (e.g. x^α). Metamath often refers to its variables as "metavariables", because they can be interpreted as variables which stand in for expressions of their type in some lower object language. Note that even vars like x are metavariables in this sense, but they can only be substituted with other vars (because the only expressions of type "var" are other vars). The capital term variables, on the other hand, can be substituted with other terms, like x^α 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

[1]The variable x is required to not be present in the (expression substituted for) R.
[2]The variable x is required to not be present in A.
[3]The variable y is required to be distinct from x and not present in B.

variable types are necessary in the Metamath interpretation. However, since this conflicts with our terminology we will stick to calling these variables, and referring to variables ranging over Metamath expressions as "metavariables". For simplicity of presentation, we will stick to using only vars during a derivation and keep the term variables to just the axioms themselves.

Most of the axioms have direct equivalents, but a few deserve extra explanation. The axioms wtru, wct, id, syl, jca, simpl, simpr, trud are necessary in order to handle context manipulation: doing the set unions in OpenTheory axioms like eqMp (this is discussed in more detail in section 2.4). The axioms hbl1, distrc, distrl, ax-17 are used to define the properties of the "not free in" predicate, which is used in inst.

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

Then ax-17 asserts that $\mathrm{NF}(x^\alpha, A)$ whenever A is a term that is distinct from x, and hbl1 asserts that $\mathrm{NF}(x^\alpha, (\lambda x^\beta.\ A))$, while distrc and distrl can be used to show that $\mathrm{NF}(x^\alpha, A)$ and $\mathrm{NF}(x^\alpha, B)$ imply $\mathrm{NF}(x^\alpha, A\ B)$ and $\mathrm{NF}(x^\alpha, (\lambda y^\beta.\ A))$ (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 `set.mm` to represent the not-free predicate, although there it is expressed more naturally as $(\varphi \to \forall x\, \varphi)$. The name ax-17 is borrowed from `set.mm`, where an axiom by the same name asserts $(\varphi \to \forall x\, \varphi)$ when x is distinct from φ.)

The three additional axioms in HOL are translated to the following three axioms:

eta $\top \vdash \forall f^{\alpha\to\beta}.\ (\lambda x^\alpha.\ f^{\alpha\to\beta}\ x^\alpha) = f^{\alpha\to\beta}$

ac $\top \vdash \forall p^{\alpha\to\mathsf{bool}}.\ \forall x^\alpha.\ (p^{\alpha\to\mathsf{bool}}\ x^\alpha \Rightarrow p^{\alpha\to\mathsf{bool}}\ (\epsilon p^{\alpha\to\mathsf{bool}}))$

inf $\top \vdash \exists f^{\mathsf{ind}\to\mathsf{ind}}.\ (\mathsf{injective}\ f^{\mathsf{ind}\to\mathsf{ind}} \wedge \neg \mathsf{surjective}\ f^{\mathsf{ind}\to\mathsf{ind}})$

These axioms are exactly what you would expect from the earlier listing on page 190. Also, the following simple theorems of the system will be useful:

$$\frac{\top \vdash A}{R \vdash A}\ \mathsf{a1i}$$

$$\frac{R \vdash A = B}{R \vdash B = A}\ \mathsf{eqcomi}$$

$$\frac{R \vdash A = B \quad R \vdash B = C}{R \vdash A = C}\ \mathsf{eqtri}$$

$$\frac{A : \mathsf{bool} \quad C : \mathsf{bool} \quad R \vdash A = B \quad R \vdash C = D}{R \vdash (A, C) = (B, D)}\ \mathsf{cteq}$$

$$\frac{\mathrm{NF}(x^\alpha, A) \quad \mathrm{NF}(x^\alpha, B)}{\mathrm{NF}(x^\alpha, (A, B))}\ \mathsf{hbct}$$

$$\frac{x^\alpha = y^\alpha \vdash A = B}{\top \vdash (\lambda x^\alpha.\ A) = (\lambda y^\alpha.\ B)}\ \mathsf{cbv}$$

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 α-equivalent), it (provably) requires eta for its `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 $\mathcal{F}$.

Definition 2.4. The map $\mathcal{F}$ 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, $\mathcal{F}(c) = c$, where a constant named c is defined in the file if it is not already present.

— For a variable term v of type α, $\mathcal{F}(v) = v^{\mathcal{F}(\alpha)}$ and $\mathcal{F}(\lambda v.\ x) = (\lambda v^{\mathcal{F}(\alpha)}.\ \mathcal{F}(x))$, where a var named v is defined in the file if it is not already present.

— For a type variable α, $\mathcal{F}(\alpha) = \alpha$, where a type variable named α is defined in the file if it is not already present.

— $\mathcal{F}(f\ x) = \mathcal{F}(f)\ \mathcal{F}(x)$

— $\mathcal{F}(t : \alpha) = \mathcal{F}(t) : \mathcal{F}(\alpha)$

— $\mathcal{F}(\alpha \to \beta) = \mathcal{F}(\alpha) \to \mathcal{F}(\beta)$

— If $A[\alpha_1, \ldots, \alpha_n]$ is a type operator of arity n, then $\mathcal{F}(A[\alpha_1, \ldots, \alpha_n]) = A[\mathcal{F}(\alpha_1), \ldots, \mathcal{F}(\alpha_n)]$, 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).

— $\mathcal{F}(\{\phi_1, \phi_2, \ldots, \phi_n\} \vdash \psi) = ((\mathcal{F}(\phi_1), \mathcal{F}(\phi_2)), \ldots, \mathcal{F}(\phi_n)) \vdash \mathcal{F}(\psi)$, unless $n = 0$ in which case $\mathcal{F}(\vdash \psi) = \top \vdash \mathcal{F}(\psi)$.

Remark 2.1. Since sets are unordered but the context conjunction is, $\mathcal{F}(\Gamma \vdash \phi)$ is not uniquely defined. However, any of the choices of ordering of Γ produce equivalent statements, and internally Γ 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. $\mathrm{NF}(x^\beta, (\lambda x^\alpha.\ x^\beta))$ even though OpenTheory would consider x^β as free in that expression). This can be ensured with suitable preprocessing 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 $\mathcal{F}(\Gamma \vdash \phi)$ 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 $\mathcal{F}$, 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 α-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 simpr instead). □

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

Proof. By induction on the length of B.

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

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

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

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

— If $B = (\lambda y^\beta.\ B_1)$ where x, y are distinct, then $C = (\lambda y^\beta.\ B_1[A/x])$ and distrl, leq, eqtri proves $\mathrm{NF}(x^\alpha, C)$. 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 $y^\beta = z^\beta \vdash B_1 = B_1[z/y]$, and then apply cbv, a1i to get $x^\alpha = A \vdash B = (\lambda z^\beta.\ B_1[z/y])$. Using the induction hypothesis once more to prove

$x^\alpha = A \vdash B_1[z/y] = C_1$, leq gives $x^\alpha = A \vdash (\lambda z^\beta.\ B_1[z/y]) = C$ and eqtri proves the goal theorem.

□

Lemma 2.3. *If A is α-equivalent to B (where α-equivalence of* `hol.mm` *terms is defined similarly to OpenTheory α-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 refl proves the goal.
— If $A = A_1\ A_2$, then $B = B_1\ B_2$ and ceq proves the goal.
— If $A = (A_1, A_2)$, then $B = (B_1, B_2)$ and cteq proves the goal.
— If $A = (\lambda x^\alpha.\ A_1)$ and $B = (\lambda x^\alpha.\ B_1)$, then leq proves the goal.
— If $A = (\lambda x^\alpha.\ A_1)$ and $B = (\lambda y^\alpha.\ B_1)$, then let $A_1' = A_1[y/x]$ and $A' = (\lambda y^\alpha.\ A_1')$. Then A is α-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 cbv gives $\top \vdash A = A'$ and eqtri proves the goal.

□

Lemma 2.4. *If A is α-equivalent to B and C is α-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 id, a1i, eqmp, eqcomi turn these into $B \vdash A$ and $C \vdash D$, and then syl gives $B \vdash D$ as desired. □

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 ϕ and a `hol.mm` term A, we say that A *reduces to* ϕ if there is a ϕ' α-equivalent to ϕ 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* substitution σ 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 Γ is equivalent to a subset of the conjunction of terms in A, and B and ϕ 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:

— varTerm → wv
— absTerm → wl
— appTerm → wc
— refl → refl
— eqMp → eqmp
— appThm → ceq
— deductAntisym → ded
— extensionality → eta
— choice → ac
— infinity → inf

Given derivations of all the hypotheses to one of these inferences, apply the transformed step to the transformed hypotheses (and a1i to the result if $A \neq \top$) to get an α-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 α-equivalent to B, followed by lemma 2.4.

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

2.4.4 *subst.* There are two kinds of substitution performed by 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 $\sigma = [x_1 \mapsto A_1, \ldots, x_n \mapsto A_n]$ is a term variable substitution, then by writing this as a composition of $[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$ with $[y_1 \mapsto A_1, \ldots, y_n \mapsto A_n]$ 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 $\sigma_1 = [x_1 \mapsto A_1], \ldots, \sigma_n = [x_n \mapsto A_n]$, so that we can reduce to the case of a single variable substitution. And this case is handled by axiom inst, with the hypotheses filled by lemma 2.2.

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

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

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

$$\frac{\top \vdash F\ B \quad \mathrm{TD}_{\beta,A,R}(F,B)}{A : \alpha \to \beta \quad R : \beta \to \alpha \quad \top \vdash (A; (R\ x^\beta) = x^\beta, F\ y^\alpha = (R\ (A\ y^\alpha) = y^\alpha))}\ \text{typedef}$$

we can set it up so that the only axiom needed for 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 $\mathrm{TD}_{\beta,A,R}(F, B)$ (which is a new kind of statement designed solely for input to typedef). However the checking that the constant does not appear in the definition, the typedef's type constant β 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. □

Proof of Theorem 2.1. Follows immediately from Remark 2.2 and Theorem 2.2. □

3. PART II: CONVERSION FROM `hol.mm` TO `set.mm`

In this part, we have the remaining job of transforming our Metamath representation of a HOL axiomatic system into ZFC. Although the foundations are changing, the basic functions of substitution and the like are the same on both the start and endpoint of the transformation, so 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:

Definition 3.1. A *type* α is a pair $\langle \iota_\alpha, b_\alpha \rangle$ of a *witness* and a *base set* such that $b_\alpha \in V_{\omega+\omega}$ (where $V_{\omega+\omega}$ is the second limit step of the cumulative hierarchy) and $\iota_\alpha \in b_\alpha$. Let $\mathrm{Ch}(\epsilon)$ denote that either ϵ is a choice function on $V_{\omega+\omega}$ or there is no such function, and define a map $\mathcal{S}_\epsilon$ from types, terms and statements of `hol.mm` to sets and wffs of `set.mm`.

— For the two constant types, we take $\mathcal{S}_\epsilon(\mathsf{bool}) = \langle 1, 2 \rangle := \langle 1, \{0, 1\} \rangle$ and $\mathcal{S}_\epsilon(\mathsf{ind}) = \langle 0, \omega \rangle$, and define $\mathcal{S}_\epsilon(\top) = 1$.

— For type variables, $\mathcal{S}_\epsilon(\mathrm{type}\,\alpha) \leftrightarrow \alpha \in \mathrm{Type}$, where $\alpha \in \mathrm{Type}$ is defined to mean that α is a type in the sense above; this hypothesis is implicit in all axioms that involve type variables.

— For convenience, define the wff predicate $\mathrm{toWff}(A) \iff A = 1$, and the function $\mathrm{toBool}(\varphi) = \mathrm{if}(\varphi, 1, 0)$.

— A variable is mapped to $\mathcal{S}_\epsilon(x^\alpha) = \mathrm{if}(x \in b_\alpha, x, \iota_\alpha)$.

— For the function type, we take $\mathcal{S}_\epsilon(\alpha \to \beta) = \langle (x \in b_\alpha \mapsto \iota_\beta), b_\beta^{b_\alpha} \rangle$, the set of all ZFC functions from α to β, 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:
$\mathcal{S}_\epsilon((A, B)) = \mathrm{toBool}(\mathrm{toWff}(A) \wedge \mathrm{toWff}(B))$.

— For a term, $\mathcal{S}_\epsilon(A : \alpha) \leftrightarrow A \in b_\alpha$.

— For a theorem, $\mathcal{S}_\epsilon(A \vdash B) \leftrightarrow \vdash (\mathrm{Ch}(\epsilon) \wedge \mathrm{toWff}(A)) \to \mathrm{toWff}(B)$.

— The equality operator $=_\alpha$ is mapped, depending on its type, to $\mathcal{S}_\epsilon(=_\alpha) = (x \in b_\alpha \mapsto (y \in b_\alpha \mapsto \mathrm{toBool}(x = y)))$

— The indefinite descriptor ϵ_α itself is mapped, depending on its type, to

$$\mathcal{S}_\epsilon(\epsilon_\alpha) = (f \in b_\alpha^{\{0,1\}} \mapsto \text{if}(\forall x \in b_\alpha.\ f(x) = 0, \iota_\alpha, \epsilon(\{x \in b_\alpha : f(x) = 1\}))).$$

— A defined type $\mathcal{S}_\epsilon(\text{TD}_{\beta,A,R}(F, B))$ 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 ϵ. 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 $V_{\omega+\omega}$, which is a set in ZFC, and thus the ZFC axiom of choice gives us a single choice function ϵ on all of $V_{\omega+\omega}$. We pass this in as a parameter to $\mathcal{S}$, so that we can give meaning to the various pieces of the formula that use ϵ, and theorems assert the choice behavior of ϵ, 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 $\vdash \mathcal{S}_\epsilon(\top \vdash A) \implies \vdash \text{toWff}(A)$ to avoid choice, so that the ZFC choice axiom never gets invoked unless ac does. Since we also require a term variable to have unconditional closure, we are forced to add witnesses so that we don't need to invoke choice by using ϵ to select elements.

The construction here is performed using $V_{\omega+\omega}$, but of course it is also possible to use V_δ for any limit ordinal $\delta > \omega$, and this can be passed in as an extra parameter to make a translation $\mathcal{S}_{\delta,\epsilon}$ which depends on δ. However, as this makes the translation process more cumbersome to describe, we will assume $\delta = \omega + \omega$ and leave the extension process to those who need the extra power this affords.

3.1 Proving the `hol.mm` axioms

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

Proof. Here we merely need to verify that each of the axioms is preserved under wrapping by $\mathcal{S}_\epsilon$. Note that $\mathcal{S}_\epsilon$ can actually be defined in `set.mm`, so that each of the axioms, in $\mathcal{S}_\epsilon$-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 α, β, 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 [col14].

— type bool: $1 \in 2 \in V_{\omega+\omega}$

— type ind: $0 \in \omega \in V_{\omega+\omega}$

— type $(\alpha \to \beta)$: If $\text{rank}(b_\alpha) = m < \omega + \omega$ and $\text{rank}(b_\beta) = n < \omega + \omega$, then $\text{rank}(b_\beta^{b_\alpha}) \le \max(m, n) + 3 < \omega + \omega$

— wtru: $1 \in 2$

— wct: toBool of anything is in 2

— wc: If $f : \alpha \to \beta$ and $x \in \alpha$, then $f(x) \in \beta$

— wv: If $x \in b_\alpha$, then $\text{if}(x \in b_\alpha, x, \iota_\alpha) = x \in b_\alpha$, otherwise $\iota_\alpha \in b_\alpha$

— wl: If for all x, $T \in b_\beta$, then $(x \in \alpha \mapsto T) : b_\alpha \to b_\beta$

— id: theorem simpr[2]

— 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 $\text{toWff}(A = B) \leftrightarrow (\text{toWff}(A) \leftrightarrow \text{toWff}(B))$ by case analysis)

— ded: theorem impbid,expr

— ceq: theorem fveq12d

— leq: theorem mpteq2dv

— hbl1: theorem hbmpt1

— distrc,distrl: short proof using vtoclg,fvmpt2

— ax-17: theorem fvmpt,eqidd

— beta: theorem fvmpt2

— inst: theorem vtoclf

— eta: theorem dffn5v

— inf: short proof using $(x \in \omega \mapsto x + 1)$

— ac: By definition, $\epsilon_\alpha(p) = \text{if}(\forall x \in b_\alpha.\ p(x) = 0, \iota_\alpha, \epsilon(\{x \in b_\alpha : p(x) = 1\}))$, but since $p(x) = 1$, the if-condition is false, so $\epsilon_\alpha(p) = \epsilon(\{x \in b_\alpha : p(x) = 1\}))$. Assuming ax-ac, ϵ is a choice function on $V_{\omega+\omega}$, and $\{x \in b_\alpha : p(x) = 1\}$ is a nonempty subset of $b_\alpha \in V_{\omega+\omega}$, so $\epsilon(\{x \in b_\alpha : p(x) = 1\}) \in \{x \in b_\alpha : p(x) = 1\}$, and thus $p(\epsilon_\alpha(p)) = 1$.

For the final statement, observe that no $\mathcal{S}_\epsilon$ transformation depends on ϵ except $\mathcal{S}_\epsilon(\epsilon_\alpha)$, so if ϵ is not in A or in B then it will not be in the right hand side of $\text{Ch}(\epsilon) \to (\mathcal{S}(A) \to \mathcal{S}(B))$, and theorem exlimiv turns this into $\exists\epsilon.\, \text{Ch}(\epsilon) \to (\mathcal{S}(A) \to \mathcal{S}(B))$. But $\exists\epsilon.\, \text{Ch}(\epsilon)$ is provable, because if there is a choice function ϵ on $V_{\omega+\omega}$ then $\text{Ch}(\epsilon)$ for that choice of ϵ and if not then $\text{Ch}(0)$ 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

[2]This and the other theorem names mentioned here refer to theorem labels in `set.mm`. There are individual web pages for these, for example simpr is found at `http://us.metamath.org/mpegif/simpr.html`.

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.

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

[col14] Metamath collaboration. Metamath Proof Explorer. `http://us.metamath.org/mpegif/mmset.html`, 2014. [Online; accessed 18-Dec-2014].

[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`.