

Sequent Style Proof Terms for HOL

Tom Ridge

LFCS, Informatics, Edinburgh University, Scotland, UK.

Abstract. In this work we present proof terms for a Gentzen sequent style presentation of HOL. Existing implementations of proof terms for HOL are natural deduction style systems. Sequent style proof terms have many advantages over natural deduction style proof terms. For example, we can translate proof terms directly into tactics, which we can execute at the tactic level of HOL implementations. We describe several applications of our work, such as an implementation of theory interpretation, and an approach to optimising proof terms by rewriting.

1 Introduction

In previous sections we have described an approach to theory interpretation based on proof term transformation, and implemented this approach in the HOL Light theorem prover. One of the problems with this implementation was that it was built on a foundation of proof terms that does not support binding, so that developing a robust theory of proof term transformation for this system appears impossible. One of the main motivations behind the work we present in the following sections is to develop a foundation of proof terms for HOL that permits reasoning about proof term transformation. However, the work stands largely on its own, and so we do not tie our presentation to the implementation of theory interpretation unnecessarily.

To ensure soundness, systems in the LCF tradition often have a type `thm` of theorems, whose constructors are tightly controlled. This isolates sources of inconsistency to the constructors of `thm`, and the routines they depend on. A `thm` is divorced from its proof, and so does not encapsulate evidence for its own correctness. On the other hand, proof terms are representations of proofs¹, and so do provide evidence for their own correctness. With proof terms one no longer has to rely on the correctness of the theorem prover which created the proof term, but can check the proof term using an independent proof checker, which is important in many applications.

Although this is the primary use of proof terms, there are many other applications. For example, imagine the following hypothetical scenario: we fix a first order language containing constants c, \dots and functions of various arities $f(x), g(x, y), \dots$, and invoke a first order prover on a given goal. The first order prover returns a proof term in some system of first order logic. We take this proof term, and transform it into a proof term in a system of higher order logic.

¹ In this work, we usually identify a proof with its representation as a proof term.

In so doing we map FOL constants and functions to HOL constants at specific types. For instance, f now has the type $\alpha \Rightarrow \beta$ for fixed constant types α, β . We take the resulting proof term, and transform it into a tactic, which we then execute at the tactic level of our HOL system against the original goal. This creates a `thm` object, and this process serves as an independent proofcheck of the proof produced by the first order prover. The proof contains many uses of *Cut*. For fun we transform the proof term using *Cut* elimination, so that it no longer contains instances of *Cut*. We have a mechanised theory of proof terms, including a formalisation of *Cut* elimination for the system of proof terms, which enables us to be certain that such transformations result in a wellformed proof term of the same theorem. We convert the proof term again into a tactic and execute it at the tactic level. Again, a `thm` object is produced, but unfortunately, the proof takes an excessive amount of time to replay. We decide to reduce the proof term size in an effort to reduce the replay time, and with the additional benefit that when we store the proof term to disk, it will take up less space. Proofs produced by automatic systems typically contain many redundant steps, and this is the case with our proof term. We remove these steps by rewriting the proof term. Again, these transformations are justified theoretically. The transformed proof term is significantly smaller. We then replay the resulting proof term at the tactic level, and find that the replay time is significantly reduced. At this point, we realise that the original proof is valid for arbitrary $c, f, g \dots$, not just the fixed $c, f, g \dots$ of the original goal. We transform the proof term by first abstracting the constants c, f, g to parameters (free variables) at the same fixed constant types. Then we abstract the constant types to type parameters. For instance, the constant f at type $\alpha \Rightarrow \beta$ for fixed constant types α, β , is now a parameter f at type $\gamma \Rightarrow \delta$, where γ, δ are arbitrary type variables. Again, we replay the proof at the tactic level, and get a `thm` object that is a general version of our original. Finally, we take the proof term and transform it into a L^AT_EX representation of the sequent level proof it represents, suitable for display. Additionally we transform the proof term into an equivalent tactic script and save it to disk.

In this work we describe a system of proof terms for HOL. This system has been implemented in HOL Light, although the system itself is applicable to all HOL implementations. The main innovation compared with previous systems of proof terms [Won95] [BN00] is that we present proof terms for a sequent system (L-style) rather than a natural deduction system (N-style). As a consequence, there are many applications that our system of proof terms support, that are either difficult or impossible to support in other systems. For instance, most proofs in HOL implementations are conducted using tactics. The tactic level presents an L-style interface to the HOL system. L-style proof terms directly capture proofs in an L-style logical system, and so have a close correspondence to primitive proofs at the tactic level. With our L-style system we can take an L-style proof term and create a tactic that we can replay against the original goal using the tactic level machinery of the HOL implementation. With an N-style proof, there is no direct relation to the L-style system that the tactic level

presents, so that it is difficult to transform a proof term into a corresponding tactic.

The contributions of this work are to describe a system of L-style proof terms, and to outline the ways in which this system can support many diverse applications. For instance, the scenario sketched above is not hypothetical, but represents the reality of our system: all the steps in the scenario have been implemented in some form. The main applications, which represent novel uses of proof terms and are supporting contributions, are an implementation of theory interpretation [Far94] by proof term transformation, and an implementation of proof optimisation by rewriting.

An overview of the paper is as follows. A main requirement for our proof terms is that they be L-style proof terms. However, the rules of existing HOL implementations are given as N-style rules. Therefore we give a standard N-style presentation of HOL, our L-style presentation, and relate the two. We then describe our basic system of proof terms. In order to support standard sequent presentations of first order logic, we then introduce a system of derived proof terms, which we justify in terms of the basic system. We mention some applications of our system and discuss the implementation in HOL Light [Har]. We discuss related work by comparing the features of our system with those of the other two main systems of proof terms for HOL. Finally we conclude and consider future work.

2 Logical System

We give the original N-style rules of HOL Light with explicit assumptions, Fig. 1, and our L-style rules for comparison, Fig. 2. Terms are the typed terms of HOL, considered modulo α -conversion. Contexts Γ are sets of boolean typed terms. Our modified rules are L-style, but because of their origin in an N-style system, we retain N-style naming conventions (intro and elim vs. left and right).

Although the two systems are superficially similar, one should keep in mind the differences between N-style systems, and L-style systems. The individual rules are also subtly different in places. The relation between these two systems is given in the following.

Theorem 1. (*Admissibility of INST_TYPE in NHOL*) *If $\Gamma \vdash C$ is derivable in NHOL, then it is derivable without the INST_TYPE rule.*

Proof. By structural induction over the derivation, from leaf to root. Essentially, *INST_TYPE* allows us to replace parametric types with concrete types. If we simply replay a proof with the parametric types replaced everywhere with the concrete types, we have a proof of the same sequent without using *INST_TYPE*.

Theorem 2. (*Admissibility of INST in LHOL*) *Any use of INST in NHOL is admissible in LHOL.*

Proof. By structural induction over the derivation, from leaf to root. Essentially because we have included full β -conversion in system LHOL, so that the rules of LHOL are closed under substitution of terms for free variables.

Theorem 3. (*Relation Between NHOL and LHOL*) $\Gamma \vdash C$ is derivable in NHOL iff $\Gamma \vdash C$ is derivable in LHOL.

$$\begin{array}{c}
\frac{}{p \vdash p} \text{ ASSUME} \\
\\
\frac{}{\vdash t = t} \text{ REFL} \qquad \frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u} \text{ TRANS} \\
\\
\frac{\Gamma \vdash p = q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q} \text{ EQ_MP} \qquad \frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p = q} \text{ DAR} \\
\\
\frac{\Gamma \vdash f = g \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash fu = gv} \text{ MKCOMB} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash \lambda x. s = \lambda x. t} \text{ ABS} \\
\\
\frac{}{\vdash (\lambda x. t)x = t} \text{ BETA} \qquad \frac{\Gamma[\bar{\alpha}_i] \vdash P[\bar{\alpha}_i]}{\Gamma[\bar{t}_i] \vdash P[\bar{t}_i]} \text{ INST} \\
\\
\frac{\Gamma[\bar{\alpha}_i] \vdash P[\bar{\alpha}_i]}{\Gamma[\bar{\gamma}_i] \vdash P[\bar{\gamma}_i]} \text{ INST_TYPE}
\end{array}$$

ABS: x does not appear free in Γ

Fig. 1. NHOL: HOL N-style Rules [Har]

$$\begin{array}{c}
\frac{}{\Gamma \vdash t = t} \text{ REFL} \qquad \frac{\Gamma \vdash s = t \quad \Gamma \vdash t = u}{\Gamma \vdash s = u} \text{ TRANS} \qquad \frac{\Gamma \vdash u = v}{\Gamma \vdash su = sv} \text{ APP_EQ} \\
\\
\frac{}{\Gamma \vdash (\lambda x. t)s = [s/x]t} \text{ BETA} \qquad \frac{\Gamma \vdash f = g}{\Gamma \vdash fu = gu} \text{ FN_EQ_E} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash \lambda x. s = \lambda x. t} \text{ FN_EQ_I} \\
\\
\frac{}{p, \Gamma \vdash p} \text{ ASSUME} \qquad \frac{\Gamma \vdash p = q \quad \Gamma \vdash p}{\Gamma \vdash q} \text{ BOOL_EQ_E} \qquad \frac{q, \Gamma \vdash p \quad p, \Gamma \vdash q}{\Gamma \vdash p = q} \text{ BOOL_EQ_I}
\end{array}$$

FN_EQ_I: x does not appear free in Γ

Fig. 2. LHOL: HOL L-style Rules

$$\begin{array}{c}
\frac{}{\Gamma \vdash t = t \triangleleft \text{refl } t} \text{REFL} \\
\frac{\Gamma \vdash s = t \triangleleft d \quad \Gamma \vdash t = u \triangleleft e}{\Gamma \vdash s = u \triangleleft \text{trans } d \ e} \text{TRANS} \\
\frac{\Gamma \vdash u = v \triangleleft d}{\Gamma \vdash su = sv \triangleleft \text{app_eq } d \ s} \text{APP_EQ} \\
\frac{}{\Gamma \vdash (\lambda x.t)s = [s/x]t \triangleleft \text{beta } s \ (\lambda x.t)} \text{BETA} \\
\frac{\Gamma \vdash f = g \triangleleft d}{\Gamma \vdash fu = gu \triangleleft \text{fn_eq_e } d \ u} \text{FN_EQ_E} \\
\frac{\Gamma \vdash s = t \triangleleft d}{\Gamma \vdash \lambda x.s = \lambda x.t \triangleleft \text{fn_eq_i } (\lambda x.d)} \text{FN_EQ_I} \\
\frac{}{h : p, \Gamma \vdash p \triangleleft \text{assume } (h : p)} \text{ASSUME} \\
\frac{\Gamma \vdash p = q \triangleleft d \quad \Gamma \vdash p \triangleleft e}{\Gamma \vdash q \triangleleft \text{bool_eq_e } d \ e} \text{BOOL_EQ_E} \\
\frac{h_1 : q, \Gamma \vdash p \triangleleft d \quad h_2 : p, \Gamma \vdash q \triangleleft e}{\Gamma \vdash p = q \triangleleft \text{bool_eq_i } (\lambda h_1 : q.d) \ (\lambda h_2 : p.e)} \text{BOOL_EQ_I}
\end{array}$$

FN_EQ_I: x does not appear free in Γ

Fig. 3. LHOL+: HOL L-style Rules, With Proof Terms

Proof. (\rightarrow) we eliminate uses of *INST_TYPE* from the NHOL derivation, then use the admissibility of *INST* in LHOL. The rule *MKCOMB* in NHOL is simulated by the two rules *FN_EQ_E* and *APP_EQ* in LHOL. We require some basic properties of the system LHOL, such as weakening, which are presented in Sect. 3. (\leftarrow) trivial. For *BETA* use *INST*.

3 Proof Terms

In Fig. 3 we present LHOL with proof term annotations, system LHOL+. Terms in context Γ are now labelled, with labels being distinct.

Theorem 4. (*Basic Properties*)

- (*Weakening*) If $\Gamma \vdash C \triangleleft d$ then $h : A, \Gamma \vdash C \triangleleft d$ where h is a new label.
- (*Strengthening*) If $h : A, \Gamma \vdash C \triangleleft d$, and h does not appear free in d , then $\Gamma \vdash C \triangleleft d$.
- (*Contraction or Hypothesis Substitution*) If $h_1 : A, h_2 : A, \Gamma \vdash C \triangleleft d$ then $h_1 : A, \Gamma \vdash C \triangleleft [h_1/h_2]d$.
- (*Term Substitution*) If $\Gamma \vdash C \triangleleft d$ and σ is a substitution of terms for free variables, then $\sigma\Gamma \vdash \sigma C \triangleleft \sigma d$.

Proof. By structural induction over the derivation of the judgements.

Definition 1. (*Wellformed*) A proof term d is wellformed iff there is some Γ and some C such that $\Gamma \vdash C \triangleleft d$.

Given a wellformed proof term d , it is easy to construct a corresponding theorem $\Gamma \vdash C$ such that $\Gamma \vdash C \triangleleft d$. Given weakening and strengthening, we should not expect such a theorem to be unique. However, C is unique, and there exists a minimum Γ , such that, if $\Gamma' \vdash C \triangleleft d$ then $\Gamma \subseteq \Gamma'$.

Theorem 5. (*Theorem Reconstruction*) Given a wellformed proof term d , there exists a minimum context Γ , and a unique boolean typed term C , such that $\Gamma \vdash C \triangleleft d$.

Proof. Structural induction over the proof term, from leaf to root. We sketch two cases.

- *trans d e*: By wellformedness, there exists Γ', s, u such that $\Gamma' \vdash s = u \triangleleft \text{trans } d \text{ e}$. So there exists t such that $\Gamma' \vdash s = t \triangleleft d$ and $\Gamma' \vdash t = u \triangleleft e$. By the induction hypothesis, we have $\Gamma_d \subseteq \Gamma'$, $\Gamma_d \vdash s = t \triangleleft d$ and $\Gamma_e \subseteq \Gamma'$, $\Gamma_e \vdash t = u \triangleleft e$. Γ' contains uniquely labelled hypotheses, so $\Gamma_d, \Gamma_e \subseteq \Gamma'$ also contains uniquely labelled hypotheses and is a wellformed context. Then Γ_d, Γ_e and $s = u$ are witnesses.
- *boolEq-i* ($\lambda h_1 : q.d$) ($\lambda h_2 : p.e$): use induction hypothesis twice to get $\Gamma_d \vdash p \triangleleft d$ and also $\Gamma_e \vdash q \triangleleft e$. Form contexts $\Gamma'_d = \Gamma_d - \{h_1\}, \Gamma'_e = \Gamma_e - \{h_2\}$. Then Γ'_d, Γ'_e and $p = q$ are witnesses.

We note that we currently reconstruct theorems in a two stage process, by first transforming a proof term into a tactic, then replaying the tactic against the original goal, see Sect. 5.

4 Derived Rules

We give an example to illustrate how the standard logical rules may be derived, and their derivations suitably abbreviated by introducing derived proof term constructors. We are going to derive the *Cut* rule.

$$\frac{h : A, \Gamma \vdash C \quad \Gamma \vdash A}{\Gamma \vdash C} \textit{Cut}$$

We introduce a constant $T = ((\lambda x.x) = (\lambda x.x))$ to stand for truth, which has the following derived rules.

$$\frac{}{\vdash T \triangleleft \textit{true}i} \textit{TRUEI} \quad \frac{\Gamma \vdash C \triangleleft d}{\Gamma \vdash C = T \triangleleft \textit{eq_T_i} d} \textit{EQ_T_I} \quad \frac{\Gamma \vdash C = T \triangleleft d}{\Gamma \vdash C \triangleleft \textit{eq_T_e} d} \textit{EQ_T_E}$$

With this in place, we construct the following derivation.

$$\frac{\frac{\frac{\vdots C}{h_1 : A, \Gamma \vdash C \triangleleft c} \quad \frac{\vdots A}{h_2 : C, \Gamma \vdash A \triangleleft a}}{\Gamma \vdash C = A \triangleleft \textit{bool_eq_i} (\lambda h_1.c) (\lambda h_2.a)} \quad \frac{\vdots A}{\Gamma \vdash A \triangleleft a}}{\Gamma \vdash C = T \triangleleft \textit{trans} (\textit{bool_eq_i} (\lambda h_1.c) (\lambda h_2.a)) (\textit{eq_T_i} a)} \quad \frac{\Gamma \vdash C = T \triangleleft \textit{trans} (\textit{bool_eq_i} (\lambda h_1.c) (\lambda h_2.a)) (\textit{eq_T_i} a)}{\Gamma \vdash C \triangleleft \textit{eq_T_e} (\textit{trans} (\textit{bool_eq_i} (\lambda h_1.c) (\lambda h_2.a)) (\textit{eq_T_i} a))}$$

This justifies the introduction of the derived proof term constructor *cut*.

$$\frac{h : A, \Gamma \vdash C \triangleleft c \quad \Gamma \vdash A \triangleleft a}{\Gamma \vdash C \triangleleft \textit{cut} a (\lambda h : A.c)} \textit{Cut}$$

Similarly we can derive all the standard left and right logical rules of a single conclusion classical² sequent system. We give these derived rules, with their associated derived proof terms, in Fig. 4. When making use of a derived rule, we can utilise these derived proof term constructors to abbreviate the derivation. Tactics use these underlying derived rules, so that these abbreviations are utilised throughout the system.

5 Applications

Standard, Extensible, L-style Proof Terms for HOL The novelty of our system lies mainly in the fact that it is an L-style system. L-style proof terms have advantages over N-style proof terms in that they are in bijection with the L-style derivation conducted at the tactic level of HOL implementations.

We have used this correspondence to create functions which *transform a proof term into a tactic*, which can be replayed against the original goal at the tactic level. This function is a simple structural pass over the proof term. Thus, given a proof term, we obtain a tactic, and having played the tactic back against the original goal, we obtain a `thm`. This allows us to reconstruct a theorem from

² Actually, rule $\neg\neg$ is only derivable if we add a classical axiom to our system so far.

$$\begin{array}{c}
\frac{}{h : A, \Gamma \vdash A \triangleleft \text{axiom } h} \text{Ax} \\
\frac{}{h : \perp, \Gamma \vdash C \triangleleft \text{false}^C h} \perp L \\
\frac{h1 : A_i, h : A_0 \wedge A_1, \Gamma \vdash C \triangleleft d}{h : A_0 \wedge A_1, \Gamma \vdash C \triangleleft \text{and}_i (\lambda h1 : A_i. d) h} \wedge L_i \\
\frac{h1 : A, \Gamma \vdash C \triangleleft d \quad h1 : B, \Gamma \vdash C \triangleleft e}{h : A \vee B, \Gamma \vdash C \triangleleft \text{orl} (\lambda h1 : A. d) (\lambda h1 : B. e) h} \vee L \\
\frac{h : A \rightarrow B, \Gamma \vdash A \triangleleft d \quad h1 : B, h : A \rightarrow B, \Gamma \vdash C \triangleleft e}{h : A \rightarrow B, \Gamma \vdash C \triangleleft \text{impl } d (\lambda h1 : B. e) h} \rightarrow L \\
\frac{h : \neg A, \Gamma \vdash A \triangleleft d}{h : \neg A, \Gamma \vdash C \triangleleft \text{notl } d h} \neg L \\
\frac{h1 : [t/x]A, h : \forall x. A, \Gamma \vdash C \triangleleft d}{h : \forall x. A, \Gamma \vdash C \triangleleft \text{foralll } t (\lambda h1 : [t/x]A. d) h} \forall L \\
\frac{h1 : [a/x]A, \Gamma \vdash C \triangleleft d}{h : \exists x. A, \Gamma \vdash C \triangleleft \text{existsl} (\lambda a. \lambda h1 : [a/x]A. d) h} \exists L^a \\
\frac{h : \neg A, \Gamma \vdash \perp \triangleleft d}{\Gamma \vdash A \triangleleft \text{negneg} (\lambda h : \neg A. d)} \neg \neg
\end{array}
\qquad
\begin{array}{c}
\frac{\Gamma \vdash A \triangleleft d \quad h : A, \Gamma \vdash C \triangleleft e}{\Gamma \vdash C \triangleleft \text{cut } d (\lambda h : A. e)} \text{Cut} \\
\frac{}{\Gamma \vdash \top \triangleleft \text{truer}} \top R \\
\frac{\Gamma \vdash A \triangleleft d \quad \Gamma \vdash B \triangleleft e}{\Gamma \vdash A \wedge B \triangleleft \text{andr } d e} \wedge R \\
\frac{\Gamma \vdash A_i \triangleleft d}{\Gamma \vdash A_0 \vee A_1 \triangleleft \text{orr}_i^{A_1 \rightarrow A_0} d} \vee R_i \\
\frac{h : A, \Gamma \vdash B \triangleleft d}{\Gamma \vdash A \rightarrow B \triangleleft \text{impr} (\lambda h : A. d)} \rightarrow R \\
\frac{h : A, \Gamma \vdash \perp \triangleleft d}{\Gamma \vdash \neg A \triangleleft \text{notr} (\lambda h : A. d)} \neg R \\
\frac{\Gamma \vdash [a/x]A \triangleleft d}{\Gamma \vdash \forall x. A \triangleleft \text{forallr} (\lambda a. d)} \forall R^a \\
\frac{\Gamma \vdash [t/x]A \triangleleft d}{\Gamma \vdash \exists x. A \triangleleft \text{existsr } t d} \exists R
\end{array}$$

$\forall R^a, \exists L^a$: a obeys standard eigenvariable condition
 $\wedge_i, \vee_i: i \in \{0, 1\}$

Fig. 4. Proof Terms For A Single-Conclusion Classical System

a proof term without having to explicitly code a reconstruction function. Of course, we have to code the function to convert the proof term into the tactic, but the process of actually executing the proof steps is left to the tactic level. In this way we save much work. The process of reconstructing a `thm` object also serves as a simple proof checker for our proof terms.

In a similar way, we have implemented a function that turns a proof term into a tactic script, such that executing the tactic script against the original goal proves that goal. This script may be included in other tactic scripts. Alternatively one can save the script to disk, giving a simple mechanism for storing proofs.

In addition, we have sequent style proof procedures which produce proof terms in our derived system that directly correspond to the steps the procedures take during proof search. Fully automatic first order theorem provers, such as resolution and tableaux provers, could also produce proof terms in the derived

system³. Using our implementation, we can easily import these proof terms into our system.

We have also created functions which transform a proof term into a sequent level proof, rendered in \LaTeX as a tree, or as a linear sequence.

These applications would be difficult with N-style proof terms, simply because there is usually no bijection between N-style proof terms, and derivations in L-style systems.

Compact Proof Terms Our system of derived proof terms is a standard L-style calculus for FOL. Automatic theorem provers usually operate in L-style systems. Thus, one can expect that the size of proof terms in the derived system, produced by automatic tactics for FOL goals, is dependent only on the size of the FOL L-style derivation.

As an example, consider an intuitionistic proof of the following variant of Pierce’s law $((((A \rightarrow B) \rightarrow A) \rightarrow A) \rightarrow \perp) \rightarrow \perp$. There is an intuitionistic proof of this theorem, in our derived system, with 11 nodes. Using our derived system with modified basic tactics, and a simple intuitionistic prover supplied with the HOL Light distribution, a proof term with size 11 is produced, as expected.

Mechanised Subsystem for First Order Logic Pfenning mechanises proofs of *Cut* elimination in Elf in [Pfe00]. To accomplish this, he first encodes first order sequent derivations as proof terms in LF. This system of sequent style FOL proof terms matches our derived system. For our derived system, Pfenning therefore provides a ready made formal framework in which to reason about metatheoretic properties of our proof terms.

Non-structural Transformation: Theory Interpretation A proof term is a tree of nodes, where each node represents an instance of a rule. By a non-structural transformation we mean a transformation that does not alter the shape of the tree, although it may alter the contents of the nodes.

Various recent papers [MB00] [JL04] transform proof terms, typically in order to reuse proofs in different settings. In fact, these techniques are all instances of theory interpretation [Far94] [Joh87].

We give a sketch of theory interpretation. The basic idea is to develop a theory axiomatically, and then transfer the results to other theories which satisfy the axioms. In this way, theory interpretation represents a primitive module system, allowing axiomatic development and reuse of theorems so proved in different contexts. For instance, one might develop a theory of groups by fixing constants to stand for the carrier of the group, and the group operations, and asserting the group axioms. Later, one realises that one needs the theorems to be instantiated for the group of automorphisms of an underlying group. This should be possible, since the automorphisms satisfy the group axioms. Ordinarily in HOL this reuse would not be possible because one had committed to an overly concrete development: the theorems hold for fixed constants and cannot

³ Actually, classical first order theorem provers will typically produce proofs in a multiple conclusion sequent system such as G3c [ST96], but the translation from G3c to our system is straight forward: one typically considers multiple conclusions as negated hypotheses.

be arbitrarily instantiated in different contexts. Theory interpretation offers a way to develop theories axiomatically, and to transfer results between different theories.

Theory interpretation can be implemented in a safe way by transforming proof terms, so that they are valid in different contexts. Theory interpretation can be founded on 3 non-structural transformations:

- Discharging axioms as assumptions.
- Changing arbitrary constants to parameters.
- Changing arbitrary fixed types to type variables.

For instance, if a proof term contains nodes that represent instances of axioms, we might wish to change these nodes into nodes that represent instances of assumptions. Instead of a proof of B using an axiom A , we now have a proof of $A \rightarrow B$ with no dependence on the axiom. These are non-structural transformations, because the shape of the proof is unchanged.

Based on these transformations, we have implemented theory interpretation conservatively in HOL Light. This allows us to surpass the modularity features found in competing systems, e.g. axiomatic classes and Locales in Isabelle. However, the downside is that proof term transformation can be expensive, and the tools are not yet ready for the non-expert user. We describe theory interpretation, and an earlier version of this work, in greater detail in [Rid05].

Structural Transformation Structural transformations are those proof term transformations that potentially alter the shape of the proof. *Cut* elimination is an example of a structural transformation. The mechanised proof of *Cut* elimination by Pfenning [Pfe00] can be directly transferred to our system, to give an algorithm that actually carries out *Cut* elimination. *Cut* elimination is a relatively complex structural transformation, so that we hope that our system will accommodate any further transformations that are dreamt up.

In this section, we describe another application of structural transformation, that appears to be a new contribution: an approach to optimising proof terms, based on higher order rewriting.

Most interactive provers employ automation to make large proofs tractable. A simple form of automation is proof search. Two main systems of proof search are resolution and tableaux. Both these systems make use of the notion of a safe rule. A safe rule is one whose application preserves provability. Thus, safe rules can typically be applied in an eager fashion. For instance, in a tableaux system, the rule $\forall L$ is usually a safe rule, whilst in an intuitionistic system, $\rightarrow L$ is usually unsafe. $\forall L$ produces two subgoals from a given goal. Thus, facing a goal $\bigvee_{i=1}^n A_i, \Gamma \vdash C$, repeated eager application of $\forall L$ results in n subgoals. If the A_i are not used in the subproofs, all we have succeeded in doing is creating n times more work for ourselves. We observe that if $h : A_i, \Gamma \vdash C \triangleleft d$, and h does not appear in d (i.e. assumption A_i was not used in the proof of the sequent) then $\Gamma \vdash C \triangleleft d$ by strengthening. Then $\bigvee_{i=0}^n A_i, \Gamma \vdash C \triangleleft d$ by weakening. In this way, a proof of a subgoal can be lifted up to a proof of the main goal. Of course, automatic theorem provers could observe that the proof of the subgoal did not

depend on the assumption, and lift the subproof up themselves. However, this rarely happens. Instead, the prover independently proves each of the subgoals. This behaviour can result in large proof terms. In fact, there is an unbounded amount of redundancy as the previous example illustrates (just increase n). For many applications, small proof terms are preferred. We now make the observation that the lifting of the subproof can be accomplished by rewriting the proof term, and that the condition that h not appear in the subproof d can be nicely captured by higher order rewriting.

$$\text{orl } (\lambda h.d) _ _ \rightsquigarrow d$$

Note that we are now using the convention that $\lambda h.d$ represents the dependence of d on the bound variable h . So $\lambda h.d$ represents that d has no dependence on h , i.e. h appears nowhere in the term d .

Our implementation of proof terms is built on top of the implementation of typed lambda terms in HOL Light. For example, our proof term constructors are represented as constants at an appropriate type. This has no logical meaning, it is simply that when looking around for an implementation of lambda terms on which to base our proof terms, we chose the system closest to hand.

With this in mind, we can represent the rewrite rule in HOL Light as follows:

```
let orl_prune1 = mk_thm([], 'orl (\a:hyp. P) Q (h:hyp)) = P');
```

We can rewrite the proof term `pft` using this rule to extract a new proof term.

```
let (_,pft') = dest_eq(concl (REWRITE_CONV [orl_prune1] pft));;
```

Since the number of redundant steps performed by automatic provers is in general unlimited, this optimisation can produce significant reductions in proof size, and consequent speed ups in proof replay. Other obvious optimising rewrites are:

```
let orl_prune1 = mk_thm([], 'orl (\a:hyp. P) Q (h:hyp)) = P');
```

```
let orl_prune2 = mk_thm([], 'orl P (\b:hyp. Q) (h:hyp)) = Q');
```

```
let andl1_prune = mk_thm([], 'andl1 (\a:hyp. P) (h:hyp)) = P');
```

```
let andl2_prune = mk_thm([], 'andl2 (\b:hyp. P) (h:hyp)) = P');
```

```
let impl_prune = mk_thm([], 'impl d1 (\a. d2) (h:hyp)) = d2');
```

We have implemented this approach to rewriting proof terms in HOL Light. In fact, because our implementation of proof terms is itself built on top of the term library of HOL Light, we can use the unmodified HOL Light rewriter to rewrite the terms directly, which saves us having to reimplement higher order matching and related routines.

This approach has some connections with program optimisation. We also note that already Nipkow investigated rewriting equational proofs in [Nip90]. Berghofer extends this work to Isabelle proof terms in [Ber03].

| | HOL4 | Isabelle | LHOL+ |
|--------------------|---------|--------------------|--------------------|
| N-style/L-style | N-style | N-style | L-style |
| Standard | N | Y | Y |
| Extensible | Y? | Y? | Y |
| Mechanised | Y? | Y (related system) | Y (derived system) |
| Binding | N | Y | Y |
| Reduction | N | Y | N |
| LF style embedding | N | N | Y |
| Non-structural | Y | Y | Y |
| Structural | N | Y | Y |
| Cut elimination | N | N | Y |

Fig. 5. Comparison of Implementations of HOL Proof Terms

6 Related Work

We compare our system with other existing implementations, whose features we tabulate in Fig. 5. In the figure, the column headed “HOL4” refers to the implementation of proof terms based on work originating in Cambridge [vW94] [Won95]. Some versions of HOL Light also use this system. The column headed “Isabelle” refers to the implementation of proof terms for Isabelle by Berghofer [BN00] [Ber03]. The last column headed “LHOL+” corresponds to the system presented here. We now give brief explanations of the feature corresponding to each row in the table.

N-style/L-style There are many ways to present logical systems. The 3 main ones are Hilbert style, Natural Deduction style (N-style), and Gentzen sequent style (L-style). These have different advantages, but L-style systems are most utilised for proof search. We provide proof terms for a single conclusion L-style implementation of HOL. Most current implementations of proof terms are N-style proof terms. Given the Curry-Howard isomorphism this is perhaps understandable: for a certain system of N-style intuitionistic logic, proofs are in bijection with typed lambda calculus terms, which can serve directly as a proof term. On the other hand, *Cut* corresponds in N-style systems to substitution of a proof tree for multiple assumptions in another proof tree. This duplication leads to very large proof trees, so that implementations of proof terms must employ subproof sharing. L-style systems support *Cut* directly, and so do not have to employ subproof sharing, leading to simple implementations. Further, most proofs are conducted (both interactively and automatically) in sequent style systems, and the tactics correspond directly to sequent style steps. At the end of a proof in N-style systems, the instructions for the sequent proof are used to produce the N-style proof term. However, this indirect link, between the system in which the proof is constructed, and the system in which the proof is represented, can lead to substantial proof term expansion. Further, if many sequent proofs correspond to a single natural deduction proof, it is impossible to reconstruct the original sequent proof from a natural deduction representation. On the other

hand, L-style proof terms can be in bijection with the tactic steps that were invoked to construct the proof. This leads to a *very direct relationship between the proofs the user constructs at the tactic level, and their representation as a proof term*. A related consideration is whether to support single conclusion, or multiple conclusion, sequents. In this we have been constrained by implementation considerations and requirements of compatibility with other libraries and tools, see Sect. 7. However, single conclusion sequent systems do not support classical logic well. It would not be difficult to adapt the approach here to a multiple conclusion sequent calculus.

Standard We aim to support the rules of first order predicate logic in a standard sequent presentation as derived rules in our system. Most current implementations of proof terms record the proof at a very low logical level, typically right at the level of the primitive rules of the system itself. On the other hand, when presenting a proof as a tree, one would like to see the standard N or L-style rules displayed. The primitive rules of HOL are not immediately related to the standard presentations of first order predicate logic. HOL is based on the notions of equality and functions, so this is to be expected. The standard logical rules are derived early on, and one typically adds corresponding derived proof term constructors which abbreviate the proof terms for the standard rules. We aim to have all the standard logical rules as derived rules, and the derived proofs represented by derived proof terms, see Sect. 4.

Extensible In Sect. 4 we give derived proof terms which extend the system and serve to abbreviate common derivations. This indicates that the system can be readily extended. In addition, we have implemented derived proof term constructors for high level operations such as **simp**. Some theorems, such as proving $\vdash m + n = p$ for large m, n, p in HOL, may have very large proofs, and this is often used as an argument in favour of incorporating β -reduction into the proof terms. However, an alternative is to add additional proof term constructors for these specific operations. Although the other systems support extension in some respects, the ease with which this can be accomplished in our system is a contribution.

Mechanised By this we mean that, not only has the system been implemented as a piece of code, but that the system itself has been mechanised inside a theorem prover, and one is capable of mechanised reasoning about aspects such as proof transformation. The HOL4 system has been mechanised, but since it does not support structural transformations adequately, this is largely irrelevant to this point. As far as we know, the Isabelle system has not been mechanised, although certainly this is an option. On the other hand, if one views the Isabelle implementation as a subsystem of pure type systems, then mechanisations do exist [MP99]. Whilst our system has not been mechanised, a certain subsystem of derived rules and proof terms for first order logic has [Pfe00], and proofs about *Cut* elimination for this system have been carried out. Our subsystem inherits this result.

Binding, Reduction and LF Embedding Existing implementations of proof terms in HOL4 and HOL Light, which merely record the invocation of

the primitive rules, do not need to support binding structures. On the other hand, transforming these proofs can prove difficult because one does not have guarantees about variable capture and so on. There are several alternatives that might be considered.

The Curry-Howard isomorphism suggests that proof terms should naturally incorporate some binding notions. Constructive type theory, and pure type systems, utilise lambda terms (more or less) directly as proof terms. Indeed, constructive type theory can be viewed as an integrated environment for handling theorems and proofs. In this setting, proofs are first class objects, and are represented by terms in the logic itself. This has implications for the implementation of proof terms for HOL based systems.

...with the advent of “pure type systems” and the λ -cube, it became clear what proof terms for λ HOL look like in principle. [BN00]

The implementation of proof terms in Isabelle follows this line. This approach is extremely elegant and has much to recommend it. These systems also take advantage of the Curry-Howard isomorphism further by demanding that β -reduction of the proof term correspond to proof normalisation. This can be useful. However, if one wishes to reason about proof terms in the same system, this has the disadvantage that β -equivalent proofs are considered identical.

...one cannot distinguish between proofs that are cut-free and proofs that are not. This is because lambda terms that are β -equal (proofs that are equal via cut-elimination) are identified. . . If one is looking for these kind of applications, it is much more promising to use the ‘coding’ of a logic in a relatively weak framework like Automath or LF. [Geu94]

Coding a logic in this way leads to an embedding of proofs in a lambda calculus like language. A fine example of this approach is Pfenning’s encoding of sequent calculi in Elf [Pfe00]. Pfenning then mechanises proofs of *Cut* elimination for these systems in an extremely elegant and natural way.

We do aim to provide formalised support for reasoning about our proof terms, in a way that distinguishes proofs that are structurally different, even if β -convertible in constructive type theory. Partly for this reason, we follow the ‘coding’ approach.

Non-structural and Structural Transformation, Cut Elimination

One of the motivations for this work was the need to be able to transform proofs in order to support an implementation of theory interpretation, see Sect. 5. An implementation of theory interpretation via proof term transformation can be founded on 3 non-structural transformations.

On the other hand, some transformations do alter the shape of the proof. For instance, in Sect. 5 we describe a process of rewriting a proof term with higher order rewrite rules, which reduces the proof size. Another example is *Cut* elimination.

It is possible to develop a theory of non-structural transformation for all 3 systems of proof terms considered here. This theory gives guarantees about the

transformation, allowing one to make complicated sequences of transformations on big proofs with confidence that a correctly formed proof term will result, and moreover that it will be a correct proof of the desired theorem.

When trying to extend these results to structural transformation one needs strong guarantees about variable binding and capture. This rules out the HOL4 implementation, which does not record binding information. However, both the Isabelle system, and our own, are capable of supporting a theory of structural proof term transformation.

Our system of derived proof terms supports *Cut* elimination. The Isabelle system is an N-style system, so there is no notion of *Cut* elimination. This should not be seen as a deficiency.

7 Implementation

We have implemented our system of proof terms for HOL in the HOL Light theorem prover. HOL Light is based on an N-style notion of proof, Fig. 1. To accommodate L-style proof terms, we derived an L-style presentation of HOL, Fig. 2. We then rebuilt HOL Light on top of these rules, whilst adding the system of proof terms in Fig. 3 underneath. The rules are presented as context sharing, which makes them suitable for backwards proof. However, HOL Light makes extensive use of derived rules, which are created in a forward fashion. Context free forms of the rules, suitable for forward reasoning, are admissible providing the contexts are compatible. We can accommodate derived rules easily by ensuring that contexts resulting from the merging of two proofs are compatible, which is largely a process of renaming hypothesis labels in contexts. We aim to make our system backwards compatible with the existing HOL Light libraries. This has largely been achieved, with substantial parts of HOL Light running unmodified with the new kernel. This is largely due to the extremely competent engineering of HOL Light in the first place, and the systematic use of abstraction layers. Although we could leave the basic HOL Light tactics mostly untouched, one of the motivations was to ensure a direct correspondence between the tactic proofs the user constructs at the sequent level, and their representation as proof terms. To this end, we modify the basic tactics so that there is a one-to-one correspondence between application of a tactic (such as `cut_tac`), and the corresponding (primitive or derived) rule. This changes are inherited by more advanced tactics. For instance, the intuitionistic tableaux prover tactic that accompanies HOL Light produces proof terms that directly correspond to the execution of the tactic at the sequent level.

The admissibility of *INST* in Thm. 2 allows one to omit *INST* from the primitive rules of LHOL, making the system more amenable to formalisation. However, a real implementation along these lines would be too inefficient. For this reason, we implement *INST* as an admissible proof term.

$$\frac{\Gamma \vdash C \triangleleft d}{[t/x]\Gamma \vdash [t/x]C \triangleleft \text{inst } t (\lambda x.d)} \text{INST}$$

INST_TYPE may be treated similarly, but with the additional complication that we must incorporate a binder ranging over types. We hope to release a public version of our system in the near future.

8 Conclusion

The proof terms described in this paper fulfil all our requirements as far as proof terms are concerned. The advantage of L-style proof terms is that many applications which are not possible in an N-style system, are easy or trivial here. However, some applications of proof terms touch other areas where there is substantial work to be done.

For example, having implemented theory interpretation, we now have a good idea of what a module system for HOL should look like. Technically, one can take many of the ideas from [Cou97]. From a user's point of view, the module system will resemble typical module systems for the programming languages ML and OCaml [Ler04]. The barrier to implementing such a system is the way context is currently handled in HOL implementations. Whereas the constructive type theory world has always been careful to provide mechanisms that are justified in terms of the underlying logic (and here we are thinking especially of the section mechanism in Coq), equivalent mechanisms are lacking in HOL implementations. In particular, the handling of constants and axioms, and packages which build on these (which includes most packages, such as the type definition package, and the datatype package) are inappropriate because they are implemented as extensions of the logical system, rather than simulated within the system. To this end, we are modifying the core of HOL Light, and hope to report on this work in a future paper.

References

- [Ber03] Stefan Berghofer. *Proofs, Programs and Executable Specifications in Higher Order Logic*. PhD thesis, Institut für Informatik, Technische Universität München, 2003.
- [BN00] Stefan Berghofer and Tobias Nipkow. Proof terms for simply typed higher order logic. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics*, volume 1869 of *LNCS*, pages 38–52. Springer, 2000.
- [Cou97] Judicael Courant. A module calculus for pure type systems. In R. Hindley, editor, *Proceedings fo the Third International Conference on Typed Lambda Calculus and Applications (TLCA '97)*, Nancy, France, 1997. Springer-Verlag LNCS.
- [Far94] W. M. Farmer. Theory interpretation in simple type theory. In J. Heering et al., editor, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer-Verlag, 1994.
- [Geu94] J.H. Geuvers. The calculus of constructions and higher order logic. *Cahiers du Centre de logique, Université catholique de Louvain, Academia, Louvain-la-Neuve (Belgium)*, 8:139–191, 1994.
- [Har] John Harrison. HOL Light. <http://www.cl.cam.ac.uk/users/jrh>.

- [JL04] Einar Broch Johnsen and Christoph Lüth. Theorem reuse by proof term transformation. In *International Conference on Theorem Proving in Higher-Order Logics TPHOLs 2004*, Lecture Notes in Computer Science. Springer, 2004.
- [Joh87] Peter T. Johnstone. *Notes on logic and set theory*. Cambridge University Press, 1987.
- [Ler04] Xavier Leroy. *Ocaml Manual*, 2004. <http://caml.inria.fr/ocaml/htmlman/index.html>.
- [MB00] Nicholas Magaud and Yves Bertot. Changing data structures in type theory: A study of natural numbers. In *Theorem Proving in Higher Order Logics, 13th International Conference, TPHOLs 2000*, volume 1869 of *LNCS*, 2000.
- [MP99] J. McKinna and R. Pollack. Some lambda calculus and type theory formalized. *Journal of Automated Reasoning*, 23(3–4), November 1999.
- [Nip90] Tobias Nipkow. Proof transformations for equational theories. In *Proc. 5th IEEE Symp. Logic in Computer Science*, pages 278–288, 1990.
- [Pfe00] Frank Pfenning. Structural cut elimination I. intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, March 2000.
- [Rid] Tom Ridge. Informatics homepage. <http://homepages.inf.ed.ac.uk/s0128214/>.
- [Rid05] Tom Ridge. Theory interpretation by proof term transformation, 2005. Published online at [Rid].
- [ST96] Helmut Schwicthenberg and A. S. Troelstra. *Basic Proof Theory*. Cambridge University Press, 1996.
- [vW94] J. von Wright. Representing higher-order logic proofs in HOL. Technical Report 323, University of Cambridge Computer Laboratory, 1994. Also in *Higher Order Logic Theorem Proving and Its Applications*, Lecture Notes in Computer Science, 859, 1994.
- [Won95] Wai Wong. Recording and checking HOL proofs. In Phillip J. Windley E. Thomas Shubert and James Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications: 8th International Workshop*, volume 971 of *Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, 1995.