A Hybrid Encoding of Howe’s Method for Establishing Congruence of Bisimilarity

A. Momigliano¹ S. J. Ambler² R. L. Crole³,⁴

Department of Mathematics and Computer Science
University of Leicester
Leicester, LE1 7RH, U.K.

Abstract
We give a short description of Hybrid, a new tool for interactive theorem proving, which was introduced in [4]. It provides a form of Higher Order Abstract Syntax (HOAS) combined consistently with induction and coinduction. We present a case study illustrating the use of Hybrid for reasoning about the lazy λ-calculus. In particular, we prove that the standard notion of simulation is a precongruence. Although such a proof is not new [5], the development is non-trivial, and we attempt to illustrate the advantages of using Hybrid, as well as some issues which are being addressed as further work.

1 Introduction and Background
This paper describes a case study in which we prove results about the meta-theory of a trivial (object level) programming language using a new mechanized tool. The programming language is very well known—it is just the untyped lazy λ-calculus. However, this language is ideal for the purposes of our case study, as we shall explain shortly. The mechanized tool, coded within Isabelle HOL, is called Hybrid; it was introduced in [4].

The key features of Hybrid are

• Hybrid provides a form of logical framework within which the syntax of an object level logic can be adequately represented by higher order abstract syntax (HOAS).

• Hybrid is compatible with tactical theorem proving in general, and principles of induction and coinduction in particular.

¹ Email:A.Momigliano@mcs.le.ac.uk
² Email:S.Ambler@mcs.le.ac.uk
³ Email:R.Crole@mcs.le.ac.uk
⁴ This work was supported by EPSRC grant number GR/M98555

©2002 Published by Elsevier Science B. V.
• It is *definitional* which guarantees consistency [12] *within a classical type theory*, while HOAS is usually associated to intuitionistic logic, e.g. Twelf [23].

The system provides a form of HOAS for the user to represent object logics. The user level is separated from the infrastructure, in which HOAS is implemented via a de Bruijn style encoding—see Section 2.

The lazy $\lambda$-calculus, introduced by Abramsky [2], is well known, and forms part of the theoretical foundations for lazy functional languages. In particular, in [5], Ambler and Crole described a simple functional programming language based on the lazy $\lambda$-calculus. In that paper, a standard operational semantics, contextual equivalence, and bisimilarity, were all encoded in Isabelle HOL. The key result was a fully mechanized proof that bisimilarity is indeed a congruence, and moreover that it coincides with contextual equivalence, following Howe’s proof technique [15]. The proof required the development of a considerable number of technical Isabelle HOL lemmas, since the encoding was strictly first-order, i.e. via de Bruijn notation.

In this paper, we replay (some of) the work described in [5], and attempt to show the utility of the Hybrid approach. This paper is very much an applied case study, and does not present any new theory. Moreover, the proof development is not complete, as it is in [5], and there are a small number of lemmas which are currently postulated rather than proved. However, the code and proofs we describe are substantial and this is our first large scale application of Hybrid. Thus we hope our development will be of use to other practitioners interested in proving properties of programming languages, and who might want to experiment with Hybrid as a higher order metalanguage.

We believe that the mechanization of a Howe’s style proof is a revealing test-case for proof assistants, more than the usual subject reduction case study, especially ones supporting some form of HOAS, since it requires:

• Support for (reasoning over) binding constructs.

• Support for proofs by structural induction, in particular over open terms (e.g. the Howe candidate relation $M \equiv N$).

• Support for proof by co-induction.

We work with program equivalences such as similarity and bisimilarity. We remind readers of the key informal ideas; for more details see for example [24]. Suppose that $s$ and $s'$ are programs, and we want to say when they have the same behaviour. A well known such relation is that of Morris-style contextual equivalence—$s$ and $s'$ are equivalent when, if inserted in any larger program fragment (context), both larger programs evaluate to the same value, or equivalently both terminate. While this notion of program equivalence is valuable and intuitive, it is indeed difficult to reason about its meta-theoretical properties, mainly due to the quantification on every possible context. *Bisimilarity* has emerged as a more manageable, yet, in this setting, equivalent idea. Roughly, $s$ and $s'$ are bisimilar if whenever $s$ evaluates to a value, so does $s'$,
and all the subprograms of the resulting values \( v \) and \( v' \) are also bisimilar, and vice versa. We will write \( s \approx^o s' \) to denote that \( s \) and \( s' \) have the same behaviour, and call the (equivalence) relation \( \text{bisimilarity} \). What is a formal definition of \( \approx^o \)? Let \( R \) be any binary relation on programs. Define a new binary relation \( \Phi(R) \) by setting \( s \Phi(R) s' \) just in case whenever \( s \Downarrow \lambda x.p \) for any \( p \), there exists a \( q \) such that \( s' \Downarrow \lambda y.q \) and for every \( r \), \( p[r/x] \) is \( R \)-related to \( q[r/y] \); analogously when \( t \Downarrow \lambda y.q \). If \( R \subseteq \Phi(R) \) (that is, \( R \) is a post fixed point) then whenever \( sRs' \) it follows that \( s \Phi(R) s' \). Thus \( R \subseteq \Phi(R) \) is a formal statement that \( s \) and \( s' \) have the "same \( R \)-evaluation behaviour". Of course \( R \) captures a formal description of one particular pattern of behaviour. We want the relation \( \approx^o \) to characterize "all possible behaviours". We can do this by taking \( \approx^o \) as the greatest relation \( R \) which is a postfixed point of \( \Phi \), that is, bisimilarity is the set coinductively defined by \( \Phi \). This yields a co-induction principle, which can be characterized by the following rule:

\[
\exists S : a \in S \quad S \subseteq \Phi(S) \quad \frac{}{a \in \text{gfp}(\Phi)} \quad CI
\]

On the other hand, establishing the equivalence of specific programs by providing the correct bisimulation can become arduous, see for example the proof of \( \text{filter } p \ (\text{map } f \ s) \approx^o \text{map } f \ (\text{filter } (p \circ f) \ s) \) in [24]. Equational reasoning would be much easier and this is why it is crucial to establish bisimulation to be a congruence.

We proceed as follows. In Section 2 we give a brief introduction to Hybrid, referring the reader to [4] for a fuller account and more examples. In Section 3 we show how the lazy \( \lambda \)-calculus is coded in Hybrid, giving a definition of bisimilarity. In Section 4 we give explanatory notes and comments on our main proof, which establishes that bisimilarity is a congruence. We finish the paper with comments on related work and concluding remarks. Note that from now on, we talk about simulations and precongruences, rather than bisimulations and congruences. Results about the latter can be obtained by symmetrizing our proofs.

In this paper we use a pretty-printed version of Isabelle HOL concrete syntax; a rule with conclusion \( C \) and premises \( H_1 \ldots H_n \) will be represented as \( \left[ H_1; \ldots; H_n \right] \Rightarrow C \). A Isabelle HOL type declaration has the form \( s :: \left[ t_1, \ldots, t_n \right] \Rightarrow t \). Isabelle HOL connectives are represented via the usual logical notation. Free variables are implicitly universally quantified. The sign \( == \) (Isabelle meta-equality) is used for equality by definition.

2 Introducing Hybrid

Hybrid has its underpinnings in the work [1] of Andrew Gordon. Gordon defines (in HOL) a de Bruijn notation in which expressions have named free variables given by strings. He can write\(^5\) \( T = \ _d\text{LAMBDA } v t \) (where \( v \) is a

---

\(^5\) The notation \( _d\text{LAMBDA} \) comes from [1]; the small “d” signifies de Bruijn notation.
Momigliano, Ambler, and Crole

(string) which corresponds to an abstraction in which \( v \) is bound in \( t \). The function \( \text{dLAMBDA} \) has a definition which converts \( T \) to the corresponding de Bruijn term; this has an outer abstraction, and a sub-term which is \( t \) in de Bruijn form, in which (free) occurrences of \( v \) are converted to bound de Bruijn indices. Gordon demonstrates the utility of this approach. It provides a good mechanism through which one may work with named bound variables, but he does not exploit the built in HOAS which HOL itself uses to represent syntax. The novelty of our approach is that we do exploit the HOAS at the meta (machine) level of Isabelle HOL.

We introduce Hybrid by example. First, some basics. Of central importance is a Isabelle HOL datatype of de Bruijn expressions, where \( \text{bnd} \) and \( \text{var} \) are the natural numbers, and \( \text{con} \) provides names for constants

\[
\text{expr} ::= \text{CON} \quad \text{VAR} \quad \text{BND} \quad \text{expr} \quad \text{ABS} \quad \text{expr}
\]

Let \( T_0 = \Lambda V_1 . \Lambda V_2 . V_1 \ V_3 \) be a genuine, honest to goodness (object level) syntax\(^6\) tree. Gordon would represent this by

\[
T_G = \text{dLAMBDA} v_1 (\text{dLAMBDA} v_2 (\text{dAPP} (\text{dVAR} v_1) (\text{dVAR} v_3)))
\]

which equals \( \text{dABS} (\text{dABS} (\text{dAPP (dBND 1) (dVAR v3)})). \) Hybrid provides a binding mechanism with similarities to \( \text{dLAMBDA} \). Gordon’s \( T \) would be written as \( \text{LAM} v . t \) in Hybrid. This is simply a definition for a de Bruijn term. A crucial difference in our approach is that bound variables in the object logic are bound variables in Isabelle HOL. Thus the \( v \) in \( \text{LAM} v . t \) is a metavariable (and not a string as in Gordon’s approach). In Hybrid we also choose to denote object level free variables by terms of the form \( \text{VAR} \ i \); however, this has essentially no impact on the technical details—the important thing is the countability of free variables. In Hybrid the \( T_0 \) above is rendered as \( T_H = \text{LAM} \ v_1 . (\text{LAM} \ v_2 . (\text{V}_1 \ \text{VAR} 3)). \) The \( \text{LAM} \) is an Isabelle HOL binder, and this expression is by definition

\[
\text{lambda} \ (\lambda v_1 . (\text{lambda} (\lambda v_2 . (v_1 \ \text{VAR} 3))))
\]

where \( \lambda v_i \) is meta abstraction and one can see that the object level term is rendered in the usual HOAS format, where \( \text{lambda} :: (\text{expr} \Rightarrow \text{expr}) \Rightarrow \text{expr} \) is a defined function, which transforms an abstraction into the “corresponding” proper de Bruijn expression. Then Hybrid will reduce \( T_H \) to the de Bruijn term \( \text{ABS} (\text{ABS (BND 1 \ \text{VAR} 3)}) \), as in Gordon’s approach. In summary, Hybrid provides a form of HOAS where object level

- free variables correspond to Hybrid expressions of the form \( \text{VAR} \ i \);
- bound variables correspond to (bound) meta variables;
- abstractions \( \Lambda V . E \) correspond to expressions \( \text{LAM} \ v . e = \text{lambda} (\lambda v . e) \);
- applications \( E_1 \ E_2 \) correspond to expressions \( e_1 \ $$ e_2 \).

Furthermore, we wish to be able to perform meta-reasoning over Hybrid

\(^6\) We use a capital \( \Lambda \) and capital \( V \) to avoid confusion with meta variables \( v \) and meta abstraction \( \lambda \).
expressions. In order to do this, we want to view the functions \textsc{CON}, \textsc{VAR}, $\&\&$, and \textsc{LAM} as data-type constructors, that is, they should be injective, with disjoint images. In fact, we identify subsets of \textit{expr} and \textit{expr} $\Rightarrow$ \textit{expr} for which these properties hold. The subset of \textit{expr} contains all those expressions which are \textit{proper}, that is a de Bruijn expression which corresponds to a \textit{\textlambda} -calculus expression. The predicate \texttt{abstr :: (expr $\Rightarrow$ expr) $\Rightarrow$ bool} encodes a subset of \textit{expr} $\Rightarrow$ \textit{expr} consisting of suitable \( e \) for which \textsc{LAM} \( v.e\ v \) is proper. Suppose that \texttt{ABS} \( e \) is proper; for example let \( e = \text{ABS} \ (\text{BND} \ 0 \ & \text{BND} \ 1) \). Then \( e \) is of level 1, and in particular there may be some bound indices which now dangle; for example \texttt{BND} \ 1 in \texttt{ABS} (\texttt{BND} \ 0 \ & \texttt{BND} \ 1). An abstraction is produced by replacing each occurrence of a dangling index with a metavariable and then abstracting the meta variable. Our example yields the abstraction \( \lambda v.\ \text{ABS} \ (\text{BND} \ 0 \ & \ v) \).

In Hybrid distinctness of the above constructors is immediate. Similar is injectivity, but for the lambda binder \textsc{LAM}, for which one can prove that is is injective on abstractions—we shall use the informal convention of writing capital letters for meta-variables which are intended to denote abstractions:

\[
[ \text{abstr} \ E; \ \text{abstr} \ F ] \implies (\textsc{LAM} v.\ E\ v = \textsc{LAM} v.\ F\ v) = (E = F)
\]

Moreover, extensionality of abstractions is provable:

\[
[ \text{abstr} \ E; \ \text{abstr} \ F; \forall i.\ E\ (\textsc{VAR} \ i) = F\ (\textsc{VAR} \ i) ] \implies E = F
\]

as is substitutivity of proper expressions in abstractions:

\[
[ \text{abstr} \ E; \ \text{proper} \ t ] \implies \text{proper} \ (E\ t)
\]

3 Coding The Lazy Lambda Calculus in Hybrid

We begin by showing how to represent the lazy \textit{\textlambda}-calculus in Hybrid. Our \textit{primary} concern will be with \textit{programs} in this calculus, that is, closed expressions. These form (as usual) a subset of the expressions given by

\[
e :: = v | \text{Fun} \ v.\ e | e_1 \ @ \ e_2 \quad \dagger \quad \\
\]

In order to render \( \dagger \) in HOAS format, using a simply typed \textit{\textlambda}-calculus with constants as metalinguage, we need constants for abstraction and application, say \texttt{cAPP} and \texttt{cABS}. Recall that in the metalinguage, application is denoted by infix $\&\&$, and abstraction by \textsc{LAM}. Then \( \dagger \) would correspond to the grammar

\[
e :: = v | \texttt{cABS} \ &\& (\textsc{LAM} v.\ E\ v) | \texttt{cAPP} \ &\& e_1 \ &\& e_2
\]

in the metalinguage. This grammar is coded in Hybrid verbatim, provided that (see Section 2) we declare \texttt{con} to consist of exactly (the names of) the two constants. We can then regard the Isabelle HOL theory Hybrid as a metalanguage which provides a form of HOAS; in particular, capture avoiding substitution is represented by meta-level $\beta$-reduction.
In order to be able to directly encode the grammar \textsuperscript{†} in Hybrid, we define \( \textit{uexp} \equiv \textit{con expr} \) and then make the following definitions

\[
\mathbin{@} : \ [ \textit{uexp}, \textit{uexp}] \Rightarrow \textit{uexp}
\]

\[
e_1 \mathbin{@} e_2 \equiv \text{CON \textit{cAPP }} e_1 \mathbin{$\&$} e_2
\]

\[
\text{Fun} . \ : \ (\textit{uexp} \Rightarrow \textit{uexp}) \Rightarrow \textit{uexp}
\]

\[
\text{Fun } x. \ e \ x \equiv \text{CON \textit{cABS }} \text{LAM } x. \ e \ x
\]

where \text{Fun} . is indeed an Isabelle HOL binder. For example, the (object level \( \lambda \)-calculus) term \( \Lambda V_1. \Lambda V_2. V_1 \ V_2 \) will be represented by \( \text{Fun } v_1. \text{Fun } v_2. \ v_1 \mathbin{$\&$} v_2 \), although the “real” underlying form is

\[
\text{CON \textit{cABS }} \mathbin{$\&$} (\text{LAM } v_1. \text{CON \textit{cABS }} \mathbin{$\&$} \text{LAM } v_2. (\text{CON \textit{cAPP }} \mathbin{$\&$} v_1 \mathbin{$\&$} v_2))
\]

We introduce some general predicates. These will be used throughout our mechanization.

- \text{isExp } e\text{ holds when the Hybrid expression } e \text{ is equal to an expression of the grammar } \textsuperscript{†}:

\[
\text{isExp } (\text{VAR } i)
\]

\[
[ \text{isExp } e_1; \text{isExp } e_2 ] \Longrightarrow \text{isExp } (e_1 \mathbin{@} e_2)
\]

\[
[ \text{abstr } E; \forall i. \text{isExp } (E \ (\text{VAR } i)) ] \Longrightarrow \text{isExp } (\text{Fun } v. \ E \ v)
\]

This is introduced because \textit{uexp} is only a type abbreviation and would not rule out exotic terms \cite{8}. One then needs to prove that this relation too is substitutive, namely:

\[
[ \text{abstr } E; \text{isExp } p; \forall i. \text{isExp } E \ (\text{VAR } i) ] \Longrightarrow \text{isExp } (E \ p)
\]

- \text{cloExp } p \text{ holds when } \text{isExp } p \text{ holds and the Hybrid expression } p \text{ has no free variables. We call such expressions \textbf{programs}, and often use the metavariables } p \text{ and } q \text{ to denote programs.}

- \text{cloAbstr } E \text{ holds when } \text{cloExp } (\text{Fun } x. \ E \ x) \text{ and moreover } \text{abstr } E \text{ holds.}

The benefits of obtaining object-level substitution via meta-level \( \beta \)-conversion are exemplified in the encoding of lazy evaluation (on closed terms) via the inductive definition of \( \Downarrow : [ \textit{uexp}, \textit{uexp}] \Rightarrow \textit{bool} \). This definition is given by the clauses

\[
\text{cloAbstr } E \Longrightarrow \text{Fun } x. \ E \ x \Downarrow \text{Fun } x. \ E \ x
\]

\[
[ p_1 \Downarrow \text{Fun } x. \ E \ x; \text{cloAbstr } E; \text{cloExp } p_2; (E \ p_2) \Downarrow v ] \Longrightarrow (p_1 \mathbin{@} p_2) \Downarrow v
\]

Standard properties such as uniqueness of evaluation and value soundness have direct proofs based only on structural induction and the introduction and elimination rules.

Our central task is to prove that applicative simulation \( \preccurlyeq : [ \textit{uexp}, \textit{uexp}] \Rightarrow \textit{bool} \) is a precongruence (Corollary 4.2). Simulation has the single coinductive
introduction rule
\[[ \text{cloExp } r; \text{cloExp } s; \\
\forall T. r \downarrow \text{Fun } x. T \ x \rightarrow (\text{cloAbstr } T \rightarrow \\
(\exists U. s \downarrow \text{Fun } x. U \ x \land \text{cloAbstr } U \land (\forall p. \text{cloExp } p \rightarrow (T \ p) \preceq (U \ p)))))] \Rightarrow r \triangleleft s\]

The HOAS style here greatly simplifies the presentation and correspondingly the meta-theory. Indeed, with the appropriate instantiation of the coinductive relation, the proof that (closed) simulation is a pre-order (and indeed bisimulation is an equivalence relation) is immediate.

4 A Commentary on the Theorem and Proofs

Recall that a precongruence is a binary relation over syntax which is a partial order that is preserved by the syntactic constructors. The definition [24] is standard and omitted. Our case study shows that Howe’s proof that simulation is a precongruence [15] can be conducted within Hybrid. Although the mathematics of the proof is essentially similar to the proof in [5], the proof itself is of fair length. We feel that it provides some evidence that our Hybrid system implements a metalanguage for HOAS over which it is possible to drive tactical reasoning involving induction and coinduction.

Howe’s technique consists of introducing another relation, the so-called Howe relation $\Gamma \vdash s \preceq^* t$, which is easily shown to be a precongruence, and then proving that it coincides with similarity. Since Hybrid is based on a traditional (co)inductive setting, the definition of the Howe relation (Table 1) involves the introduction of open terms. We note that in a framework which fully supports HOAS [22], such as Edinburgh LF, this could be attained using a hypothetical judgment on closed terms only, where the clauses for variables and functions are merged in the following:

$\forall y. (\forall m. y \preceq m \rightarrow y \preceq^* m) \rightarrow N \ y \preceq^* N' \ y$ \quad Fun \ v. N' \ v \preceq q$

Alas, such an introduction rule yields a non-monotone operator and it is therefore disallowed in Isabelle HOL. Thus, we are lead to defining open similarity over open expressions, which is the obvious extension of similarity where two open terms are related if their $\lambda$-closures are similar. One then proves that open similarity is a precongruence. Although this corresponds to the informal mathematical development, it might seem rather indirect at first sight.

One way to encode definitions over open terms is via judgments that relate environments (lists) of free variables to expressions. In particular, we introduce inductive definitions for well-formedness of environments, i.e. no repetitions, ($\Gamma \vdash$) and of expressions ($\Gamma \vdash t$), i.e. such that all the free variables of the term $t$ occur in a well-formed $\Gamma$. Some lemmas ensure the mutual consistency
of those two notions.

Then, open simulation $\Gamma \vdash s \preceq^0 t$ is inductively defined by the rules

$$s \preceq t \implies \emptyset \vdash s \preceq^0 t$$

$$[\text{isExp } s; \text{isExp } t; n \notin \Gamma; \forall p. \text{cloExp } p \longrightarrow \Gamma \vdash \text{subst } s n p \preceq^0 \text{subst } t n p] \implies \Gamma, n \vdash s \preceq^0 t$$

Notice that in the definition the replacement in $s$ of VAR $n$ with $p$ is implemented with a primitive notion of substitution, at the de Bruijn level. Clearly, it would be highly desirable to utilize HOAS to express this clause, making use of the notion of abstraction, but the above formulation has shown to be more amenable to mechanization. The following properties are then proved:

- Open simulation preserves well-formedness of expressions and environments.
- Simulation is included in open simulation, by induction on the structure of $\Gamma \vdash$.
- Open simulation is a pre-order, by list induction.
- It is substitutive, whose proof needs the following generalization:

$$[\Gamma_1, n, \Gamma_2 \vdash s \preceq^0 t; \Gamma_1, \Gamma_2 \vdash p] \implies \Gamma_1, \Gamma_2 \vdash \text{subst } s n p \preceq^0 \text{subst } t n p$$

The proof is a somewhat lengthy list induction, utilizing certain low-level context properties, such as that open simulation is preserved by permuting distinct elements in a context.

The key theorem which we have verified using Hybrid is

**Theorem 4.1** The relation of open similarity, $\Gamma \vdash s \preceq^0 t$, is a precongruence.

An immediate corollary is

**Corollary 4.2** The relation of similarity, $s \preceq t$, is a precongruence.

That bisimilarity is a congruence follows simply from this. The idea of the corollary is that, once proved, one may reason about similarity of programs using the usual rules of algebraic reasoning.

Before giving the proof outline, we inductively introduce the Howe relation which has type $\preceq^* : [\text{var list, uexp, uexp}] \Rightarrow \text{bool}$, in Table 1.

It is immediate that this relation is a precongruence, since its definition is structural and (open) similarity is reflexive. Thus we can prove the theorem if we can demonstrate that the Howe relation and open similarity coincide. Here is the structure of this proof.

(i) We prove some general properties of the Howe relation. These are:

(a) The composition of the Howe relation with open similarity is contained in Howe. Here we present an illustrative sample of one of our simplest proof scripts in Table a; *howe.induct* in the structural
\( \Gamma \vdash \text{VAR} \, x \preceq^g \, m \implies \Gamma \vdash \text{VAR} \, x \preceq^* \, m \)

\[
\forall y. \Gamma, y \vdash N (\text{VAR} \, y) \preceq^* N' (\text{VAR} \, y);
\]

\[
\text{abstr} \, N; \text{abstr} \, N'; \Gamma \vdash \text{Fun} \, v. N' \, v \preceq^g \, q \implies \Gamma \vdash \text{Fun} \, v. N \, v \preceq^* \, q
\]

\[
\begin{align*}
[ \Gamma \vdash m_1 & \preceq^* m_1'; \Gamma \vdash m_2 \preceq^* m_2'; \Gamma \vdash (m_1' @ m_2') \preceq^g \, n ] \\
& \implies \Gamma \vdash (m_1 @ m_2) \preceq^* \, n
\end{align*}
\]

Table 1
Definition of \( \Gamma \vdash s \preceq^* \, t \)

induction principle generated by the Howe’s relation inductive definition, \texttt{hoe.intrs} denotes its introduction rules, and \texttt{best_tac} is Isabelle’s automatic best-search tactic. The command \texttt{addDs} instructs the prover to use the lemma \texttt{opensim_trans} in a forward-chaining way.

(b) Howe is reflexive, that is \( \Gamma \vdash s \implies \Gamma \vdash s \preceq^* \, s \). This is proven by induction on the structure of the antecedent, using reflexivity of open similarity and properties of well-formed expressions.

(c) Open similarity is contained within Howe, which follows immediately from (a) and (b).

(d) The Howe relation is substitutive: formally

\[
\begin{align*}
\Gamma, y \vdash s & \preceq^* \, s' & \Gamma \vdash t \preceq^* \, t' \\
\Gamma \vdash s[t/y] & \preceq^* \, s'[t'/y]
\end{align*}
\]

Note that this lemma is fundamental and is still needed in a full HOAS account\(^7\) (while substitutivity of simulation is not). Here, again, the statement needs generalizing to:

\[
\begin{align*}
[ \Gamma_1, y, \Gamma_2 & \vdash s \preceq^* \, s'; \Gamma_1, \Gamma_2 \vdash t \preceq^* \, t' ] \\
& \implies \Gamma_1, \Gamma_2 \vdash \text{subst} \, s \, y \, t \preceq^* \, \text{subst} \, s' \, y \, t'
\end{align*}
\]

The proof, the longest of our script, is by induction on the derivation of the first judgment, using substitutivity of open similarity. Since

---

\(^7\) cf. the analogous lemma about substitutivity of parallel reduction in the Twelf proof of the Church-Rosser theorem \cite{25}, which has a remarkable fully automatic proof from first principles.

Goal "env |- s <howe> t ==> 
      env |- t <opensim> u --> env |- s <howe> u";
be howe.induct 1;
by(ALLOGOALS(best_tac(HOL_cs addIs howe.intrs 
                        addDs [opensim_trans])));
qed_spec_mp "howe_semitrans";

Table 2
Proof script for semi-transitivity of the Howe relation.

68
the latter is defined via the Hybrid primitive notion of substitution, namely subst, our mechanization is rather convoluted and ad hoc, using several Hybrid infra-structural properties concerning the interaction between abstractions and substitution.

(ii) If \( \not\vdash \text{Fun}\,x.\,E\,x \preccurlyeq^* p \), then \( p \Downarrow \text{Fun}\,y.\,F\,y \) where \( \text{cloAbstr}\,F \) and for every closed \( p' \) we have \( \not\vdash E\,p' \preccurlyeq^* F\,p' \). This is proven first by inversion on the Howe relation, open similarity and eventually similarity; note that when using elimination rules, injectivity of Fun, which follows from its definition in terms of LAM, is of crucial use. Finally, the result follows from semi-transitivity and (an instance of) substitutivity of Howe.

(iii) If \( \not\vdash p \preccurlyeq^* q \) and \( p \Downarrow v \), then \( \not\vdash v \preccurlyeq^* q \). The proof closely follows the informal one, involving an induction on evaluation, and inversion on Howe and (open) simulation, with an additional case analysis on \( v \). Nevertheless, due to a fair amount of forward chaining, the degree of automation is relatively low.

Once all of these properties have been proved, establishing the main theorem by showing coincidence of the Howe relation and open similarity is easy.

\textbf{Proof.} (Of Theorem 4.1) We only need to show that Howe is contained in open similarity, point (c) above establishing the opposite containment. Recall that in order to prove \( \Gamma \vdash e_1 \equiv^\circ e_2 \), we have to prove that the \( \lambda \)-closures of \( e_1 \) and \( e_2 \) are related by closed simulation. However, having proved that the Howe relation is substitutive, this will follow if we can prove that

\[ \not\vdash p \preccurlyeq^* q \implies \not\vdash p \equiv^\circ q \]

Similarity is coinductively defined, and thus we simply have to check that \( \not\vdash p \preccurlyeq^* q \) is indeed a simulation. Suppose that \( p \Downarrow \text{Fun}\,x.\,T\,x \). By (3) we have \( \not\vdash \text{Fun}\,x.\,T\,x \preccurlyeq^* q \) and by (2) we have \( q \Downarrow \text{Fun}\,y.\,U\,y \) where \( \text{cloAbstr}\,U \) and for all closed \( p' \) we have \( \not\vdash T\,p' \preccurlyeq^* U\,p' \). We are done.

\[ \square \]

5 Related Work

Here we only review papers that use some form of HOAS to encode proofs about (bi)similarity; we refer, for example, to [4] for a review of more general issues related to HOAS and (co)induction; we just mention [11] as an early case study utilizing a first order approach.

The only other comparable mechanized proof about bisimulation in the lazy \( \lambda \)-calculus, to the best of our knowledge is [21]. This follows the Weak HOAS [22] approach (i.e. object-level substitution is encoded as an inductive relation) supplemented with the Theory of Contexts, [13]; the latter consists of a set of axioms, parametric to the HOAS signature, including the reification of key properties of names akin to freshness and, more crucially, higher-order induction and recursion schemata. The author proves that bisimulation co-
incides with observational equivalence not via Howe’s method, but following Stoughton’s adaptation of Milner’s context lemma reported in [3]. This approach may become more unwieldy in a more expressive functional language; moreover, the formalization heavily relies on vectors of terms, to encode a linearized notion of bisimulation, viz. \( M_1 \preceq M_2 \) iff for every vector \( \vec{N} \) if \( M_1\vec{N} \downarrow \) then \( M_2\vec{N} \downarrow \). Although conceptually non-problematic, several steps have not been fully verified yet (Marino Miculan, personal communication).

The Weak HOAS approach has been more successful in the context of the \( \pi \)-calculus [14]. The latter paper contains formal verification, among others, of strong late bisimilarity being a congruence. This can arguably be attributed not only to the possibility to “reflect” on names which is crucial for the meta-theory of operations such as mismatch, but also because here hypothetical judgments, which are only partially supported in such a style [8], are typically not needed. In fact, the constructors for the type of processes do not give rise to any negative occurrences of the type. Therefore \( \beta \)-conversion can implement object-level substitution, which in this case is simply “name” for bound variable in a process.

In [19] the authors present an encoding of CCS in \( FO\lambda^\Delta^I N \). Co-inductive reasoning is simulated with a representation of bisimulation via induction on natural numbers; this exploits the co-continuity of the said notion, which allows one to capture the greatest fix point via the intersection of all powers \( n \leq \omega \) starting with the universal relation. Some congruence properties have been proof-checked with the \( Pi \) editor [9]. It is possible to pursue a similar approach in a system such as Twelf [23].

We conclude with a brief comparison with the proof in [5]; as we mentioned the syntax was represented using a de Bruijn style notation, with the usual loss of human readability. Bound variable substitution was coded directly within the Isabelle HOL theory which represents the calculus. To partially alleviate this, closed simulation was defined via \( \beta \) expansion, which, in the notation of this paper, would read

\[
\left[ \text{cloExp } r; \text{cloExp } s; \forall t. r \downarrow t \right] \rightarrow \exists u. s \downarrow u \land (\forall p. \text{cloExp } p \rightarrow (t \otimes p) \preceq (u \otimes p)) \Rightarrow r \preceq s
\]

The encoding of open similarity and the Howe relation were essentially analogous to those of this paper, with the notable exception that the substitution predicate in the Hybrid encoding belongs to the infrastructure rather than the object logic.

6 Conclusions and Future Work

In this paper we have reported our experience in using Hybrid to machine-check a non-trivial result about program equivalence in the lazy \( \lambda \)-calculus.
Here are some of the lessons we have learned:

(i) The current version of Hybrid uses the predicate `abstr` to characterize the functions $E :: expr \Rightarrow expr$ which could legitimately form the body of a lambda-term $\text{LAM} x. E \ x$. The principle of induction for such abstractions introduces the notion of a ‘bi-abstraction’: a function $f :: expr \Rightarrow expr \Rightarrow expr$ which could form the body of a lambda-term $\text{LAM} x. \text{LAM} y. f \ x \ y$. This process continues indefinitely, with functions of three, four, five arguments, and so on. It would be better to have a general theory of $N$-ary abstractions and induction over these. This would reduce the amount of infra-structure, subsuming the theories which describe abstraction and bi-abstraction in one go. We have been working on such an extension and already have a reasonable prototype.

(ii) A related issue, which arises again and again throughout this work, is how to handle the ‘unbinding’ of a variable in an expression. This occurs first in the definition of $\text{isExp}$ in the clause

$$[\text{abstr} \ E; \forall i. \text{isExp} (E \ (\text{VAR} \ i)) ] \Rightarrow \text{isExp} (\text{Fun} \ v. E \ v)$$

Thus, $\text{Fun} x. E \ x$ is an expression of the object language if the body $E$ is an abstraction and moreover, every instantiation $E \ (\text{VAR} \ y)$ of the body by a variable is an object expression. The question is how should $y$ be quantified? We might have ‘all $y$', ‘all fresh $y$', ‘some fresh $y$' or, in this case, even ‘some $y$’. Each of these gives rise to a different principle of induction for expressions. They can be proved equivalent using techniques presented by McKinna and Pollack [20], but to do so is a great deal of work and apparently has to be repeated for each judgment which is defined inductively. The quantifier above is essentially trying to capture the $\text{NEW}$ quantifier of Gabbay and Pitts [16]. This is jointly universal and existential in nature. It is an open question how to best represent the dual nature of the $\text{NEW}$ quantifier in Hybrid. One possibility is to provide a uniform framework for defining judgments inductively and to prove the equivalence of ‘for all fresh’ and ‘for some fresh’ once-and-for-all using the McKinna and Pollack technique.

(iii) The infrastructure provided by Hybrid is too stratified at the moment. Many of the lemmas which are proved at the level of Hybrid have analogues at are repeated at the object level. A better approach would be to discard the $\text{proper}$ predicate and work instead with a generic predicate ‘$\text{isExp}$’ which is parameterized with the binding signature of the object logic. This should reduce the burden of administrative lemmas.

(iv) While the tool seems successful in providing a form of HOAS when dealing with closed terms, we had to resort to a more traditional encoding, i.e. via explicit environments, with respect to judgments involving open ones such as the Howe relation. This is due to the fundamental incompatibility of non-stratified hypothetical judgments and (co)induction, and
it is intrinsically problematic. Nevertheless, some other avenues can be explored.

- We could work directly on open terms, so that similarity need not be presented in two flavors. The main difficulty is not just its oddity w.r.t. evaluation in a functional programming setting, but that it seems that such a notion of similarity is not known in the literature. In fact, the related concept of weak head normal form simulation [17] coincides with Lévy-Longo tree equivalence—not applicative bisimulation.

- We may choose to embrace the two-level approach of Miller & McDowell [18], where specification and reasoning on an object logic is done in the same system but at different levels. In particular, Felty has proposed [10] a realization of the latter in Coq, where the rule of definitional reflection of $\text{FO}\lambda\Delta^N$ is mimicked by the elimination rules of inductive types supplemented by a set of axioms stating the freeness properties of constructors of the given signature. An intriguing possibility we are currently investigating is to use Hybrid in place of a system such as Coq as the meta-meta-logic for the latter architecture; this has several advantages:
  · Freeness of constructors and more importantly extensionality properties at higher types are not assumed, but proven via the related properties of the infrastructure.
  · Only non stratifiable hypothetical judgments, such as typing, needs to be encoded at the object-level, while the rest can live as a honest-to-goodness Isabelle HOL inductive definition; this will make the more mechanizable structural induction available, as compared to complete induction on heights of derivations in the specification logic [18,10].
  · Coinductive principles are available, possibly expressed at the meta-level and then reflected at the object one. This is in contrast with $\text{FO}\lambda\Delta^N$, which, in the current formulation, allows only the somewhat awkward inductive encoding via greatest fix points.
  · The specification logic does not have to be the classical one provided by HOL, but can be varied to be, for example, a fragment of linear logic. This would allow the utilization of the most elegant encodings of the meta-theory of functional programming with references proposed in [7,18].

Indeed, formal verification of the compiler optimization transformations for Benton & Kennedy’s MIL-lite language [6] is one of our main objectives.

Source files for the Isabelle HOL code can be found at

http://www.mcs.le.ac.uk/mechsem/Hybrid/Howe
References


