A Hybrid Encoding of Howe's Method for Establishing Congruence of Bisimilarity. (Extended Abstract)

A. Momigliano (A.Momigliano@mcs.le.ac.uk)
S. J. Ambler (S.Ambler@mcs.le.ac.uk) &
R. L. Crole (R.Crole@mcs.le.ac.uk)

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 automated 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 will be 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.
- It is *definitional* which guarantees consistency *within a classical type theory* [12].

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 λ-calculus, introduced by Abramsky [2], is of course extremely 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 λ-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. 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. Due to space restrictions, we shall assume that readers are familiar with program equivalences such as bisimilarity; see for example [22]. We refer to [4] for a full account of Hybrid and more examples. We proceed as follows. In Section 2 we give a brief introduction to Hybrid. In Section 3 we show how the lazy λ-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 $[\![H_1; \ldots; H_n]\!] \Longrightarrow C$. A Isabelle HOL type declaration has the form $s :: [t_1, \ldots t_n] \Rightarrow t$. 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[1] $T = \mathsf{dLAMBDA}\ \mathtt{v}\ t$ (where v is a string) which corresponds to an abstraction in which v is bound in t. The function 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 *bnd* and *var* are the natural numbers, and *con* provides names for constants

[1] The notation dLAMBDA comes from [1]; the small "d" signifies de Bruijn notation.

$$expr ::= \mathsf{CON}\ con \mid \mathsf{VAR}\ var \mid \mathsf{BND}\ bnd \mid expr\ \$\$\ expr \mid \mathsf{ABS}\ expr$$

Let $T_O = \Lambda V_1.\, \Lambda V_2.\, V_1\ V_3$ be a genuine, honest to goodness (object level) syntax[2] tree. Gordon would represent this by

$$T_G = \mathsf{dLAMBDA\,v1\,(dLAMBDA\,v2\,(dAPP\ (dVAR\,v1)\ (dVAR\,v3)))}$$

which equals dABS (dABS (dAPP (dBND 1) (dVAR v3))). Hybrid provides a binding mechanism with similarities to dLAMBDA. Gordon's T would be written as $\mathsf{LAM}\, v.\, t$ 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 $\mathsf{LAM}\, v.\, t$ 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 $\mathsf{VAR}\ i$; however, this has essentially no impact on the technical details—the important thing is the countability of free variables. In Hybrid the T_O above is rendered as $T_H = \mathsf{LAM}\, v_1.\, (\mathsf{LAM}\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))$. The LAM is an Isabelle HOL binder, and this expression is by *definition*

$$\mathsf{lambda}\ (\lambda\, v_1.\, (\mathsf{lambda}\ (\lambda\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))))$$

where λv_i is meta abstraction and one can see that the object level term is rendered in the usual HOAS format, where $\mathsf{lambda} :: (expr \Rightarrow expr) \Rightarrow expr$ 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 ABS (ABS (BND 1 $$ 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 $\mathsf{VAR}\ i$;
- bound variables correspond to (bound) meta variables;
- abstractions $\Lambda V.\, E$ correspond to expressions $\mathsf{LAM}\, v.\, e == \mathsf{lambda}\ (\lambda\, v.\, e)$;
- applications $E_1\ E_2$ correspond to expressions $e_1\ \$\$\ e_2$.

Furthermore, we wish to be able to perform meta-reasoning over Hybrid expressions. In order to do this, we want to view the functions CON, VAR, $$, and LAM. as data-type constructors, that is, they should be injective, with disjoint images. In fact, we identify subsets of *expr* and $expr \Rightarrow expr$ for which these properties hold. The subset of *expr* consists of those expressions which are *proper*, that is a de Bruijn expression which corresponds to a λ-calculus expression. The subset of $expr \Rightarrow expr$ consists of all those e for which $\mathsf{LAM}\, v.\, e\, v$ is proper. This is coded by the predicate $\mathsf{abstr} :: (expr \Rightarrow expr) \Rightarrow bool$. Suppose that $\mathsf{ABS}\ e$ is proper; for example let $e = \mathsf{ABS\ (BND\ 0\ \$\$\ BND\ 1)}$. Then e is of level 1, and in particular there may be some bound indices which now dangle; for example BND 1 in ABS (BND 0 $$ 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 $\lambda\, v.\, \mathsf{ABS\ (BND\ 0\ \$\$}\ v)$.

[2] We use a capital Λ and capital V to avoid confusion with meta variables v and meta abstraction λ.

In Hybrid distinctness of constructors is immediate, while one can *prove* injectivity of abstractions:

$$[\![\, \mathsf{abstr}\ e;\ \mathsf{abstr}\ f \,]\!] \Longrightarrow (\mathsf{Lam}\, x.\ e\ x = \mathsf{Lam}\, y.\ f\ y) = (e = f)$$

Moreover, extensionality of abstractions is also provable:

$$[\![\, \mathsf{abstr}\ e; \mathsf{abstr}\ f; \forall i.\, e\ (\mathsf{VAR}\ i) = f\ (\mathsf{VAR}\ i) \,]\!] \Longrightarrow e = f$$

3 Coding The Lazy Lambda Calculus in Hybrid

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

$$e ::= v \mid \mathsf{Fun}\, v.\, e \mid e_1\ @\ e_2 \qquad \dagger$$

In order to render † in HOAS format, using a simply typed λ-calculus with constants as metalanguage, we need constants for abstraction and application, say $cAPP$ and $cABS$. Recall that in the metalanguage, application is denoted by infix \$\$, and abstraction by LAM. Then † would correspond to the grammar

$$e ::= v \mid cABS\ \$\$\ (\mathsf{LAM}\, v.\, e\, v) \mid cAPP\ \$\$\ e_1\ \$\$\ e_2$$

in the metalanguage. This grammar is coded in Hybrid verbatim, provided that (see Section 2) we declare *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 β-reduction.

In order to be able to directly encode the grammar † in Hybrid, we define $uexp = con\ expr$ and then make the following definitions

$$@\ ::\ [\, uexp, uexp\,] \Rightarrow uexp \qquad \mathsf{Fun}.\ ::\ (uexp \Rightarrow uexp) \Rightarrow uexp$$
$$e_1\ @\ e_2 = \mathsf{CON}\ cAPP\ \$\$\ e_1\ \$\$\ e_2 \qquad \mathsf{Fun}\, x.\, e\ x = \mathsf{CON}\ cABS\ \$\$\ \mathsf{LAM}\, x.\, e\ x$$

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

$$\mathsf{CON}\ cABS\ \$\$\ (\mathsf{LAM}\, v_1.\, \mathsf{CON}\ cABS\ \$\$\ \mathsf{LAM}\, v_2.\, (\mathsf{CON}\ cAPP\ \$\$\ v_1\ \$\$\ v_2)$$

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

- isExp e holds when the Hybrid expression e is equal to an expression of the grammar †. This is introduced because *uexp* is only a type abbreviation and would not rule out exotic terms.

- cloExp p holds when isExp p holds and the Hybrid expression p has no free variables. We call such expressions **programs**, and often use the meta-variables p and q to denote programs.
- cloAbstr E holds when cloExp (Fun $x.\, E\ x$) and moreover abstr E holds. (We shall use the informal convention of writing capital letters for meta-variables which are intended to denote abstractions.)

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

$$\mathsf{cloAbstr}\ E \Longrightarrow \mathsf{Fun}\, x.\, E\ x \Downarrow \mathsf{Fun}\, x.\, E\ x$$
$$[\![\, p_1 \Downarrow \mathsf{Fun}\, x.\, E\ x;\ \mathsf{cloAbstr}\ E;\ \mathsf{cloExp}\ p_2;\ (E\ p_2) \Downarrow v\,]\!] \Longrightarrow (p_1\ @\ p_2) \Downarrow v$$

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 $\preccurlyeq\ :: [\, uexp, uexp\,] \Rightarrow bool$ is a precongruence (Corollary 1). Simulation has the single coinductive introduction rule

$$[\![\, \mathsf{cloExp}\ r;\ \mathsf{cloExp}\ s;\ \forall T.r \Downarrow \mathsf{Fun}\, x.\, T\ x \longrightarrow (\mathsf{cloAbstr}\ T \longrightarrow$$
$$(\exists U.s \Downarrow \mathsf{Fun}\, x.\, U\ x \wedge \mathsf{cloAbstr}\ U \wedge (\forall p.\ \mathsf{cloExp}\ p \rightarrow\ (T\ p) \preccurlyeq\ (U\ p))))\,]\!] \Longrightarrow\ r \preccurlyeq\ s$$

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 [22] 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 $\Gamma \vdash\ s \preccurlyeq^{\bullet}\ t$, 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. However, in a framework such as LF, the same could be attained using a hypothetical judgment on closed terms, where the clauses for variables and functions are merged in the following:

$$\frac{\forall y.(\forall m.\ y \preccurlyeq\ m \longrightarrow\ y \preccurlyeq^{\bullet}\ m) \longrightarrow\ N\, y \preccurlyeq^{\bullet}\ N'\, y \qquad \mathsf{Fun}\, v.\, N'\, v \preccurlyeq\ q}{\mathsf{Fun}\, v.\, N\, v \preccurlyeq^{\bullet}\ q}$$

Alas, such an introduction rule yields a non-monotone operator and it is therefore disallowed. Thus, we are lead to defining *open similarity* on open expressions, which is the obvious extension of similarity where two open terms are related if their λ-closures are similar. One then proves that open similarity is a precongruence. Although this corresponds to the informal mathematical development, it involves the duplication of some work.

One way to encode definitions on open terms is via judgments that relate environments (lists) of free variables to expressions, such that the free variables of the term t all occur in Γ. In particular, open simulation is inductively defined by the rules

$$[\![\ \mathsf{cloExp}\ s;\ \mathsf{cloExp}\ t;\ \ s \preccurlyeq t\ ]\!] \Longrightarrow [] \vdash\ s \preccurlyeq^{\circ}\ t$$
$$[\![\ \mathsf{isExp}\ s;\ \mathsf{isExp}\ t;\ \forall p.\ \mathsf{cloExp}\ p \longrightarrow \Gamma \vdash\ \mathsf{subst}\ s\ n\ p \preccurlyeq^{\circ}\ \mathsf{subst}\ t\ n\ p\ ]\!] \Longrightarrow \Gamma, n \vdash\ s \preccurlyeq^{\circ}\ t$$

Notice that in the definition the replacement in s of $\mathsf{VAR}\ n$ 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 key theorem which we have verified using Hybrid is

Theorem 1. *The relation of open similarity, $\Gamma \vdash\ s \preccurlyeq^{\circ}\ t$, is a* ***precongruence.***

An immediate corollary is

Corollary 1. *The relation of similarity, $s \preccurlyeq t$, 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 $[\,var\ \mathsf{list}, uexp, uexp\,] \Rightarrow bool$, in Table 1.

$$\Gamma \vdash\ \mathsf{VAR}\ x \preccurlyeq^{\circ}\ m \Longrightarrow \Gamma \vdash\ \mathsf{VAR}\ x \preccurlyeq^{\bullet}\ m$$
$$[\![\ \forall y.\Gamma, y \vdash\ N\,(\mathsf{VAR}\ y) \preccurlyeq^{\bullet}\ N'\,(\mathsf{VAR}\ y);\ \mathsf{abstr}\ N; \mathsf{abstr}\ N'; \Gamma \vdash\ \mathsf{Fun}\ v.\ N'\ v \preccurlyeq^{\circ}\ q\ ]\!] \Longrightarrow \Gamma \vdash\ \mathsf{Fun}\ v.\ N\ v \preccurlyeq^{\bullet}\ q$$
$$[\ \Gamma \vdash\ m_1 \preccurlyeq^{\bullet}\ m_1'; \Gamma \vdash\ m_2 \preccurlyeq^{\bullet}\ m_2'; \Gamma \vdash\ (m_1' \mathbin{@} m_2') \preccurlyeq^{\circ}\ n\ ] \Longrightarrow \Gamma \vdash\ (m_1 \mathbin{@} m_2) \preccurlyeq^{\bullet}\ n$$

Table 1. Definition of $\Gamma \vdash\ s \preccurlyeq^{\bullet}\ t$

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.

1. 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 our code in Table 2.
   (b) Howe is reflexive, that is $\Gamma \vdash s \Longrightarrow \Gamma \vdash s \preccurlyeq^\bullet s$. This is proven by a simple induction on the structure of the judgment $\Gamma \vdash s$ using reflexivity of open similarity.
   (c) Open similarity is contained within Howe, which follows immediately from (a) and (b).
   (d) The Howe relation is substitutive: formally

$$\frac{\Gamma, y \vdash s \preccurlyeq^\bullet s' \qquad \Gamma \vdash t \preccurlyeq^\bullet t'}{\Gamma \vdash s[t/y] \preccurlyeq^\bullet s'[t'/y]}$$

   Note that this lemma is crucial and would still be needed in a full HOAS account (cf. the analogous lemma about substitutivity of parallel reduction in the Twelf proof of the Church-Rosser theorem [23]). The proof is by induction on the derivation of the first judgment, using substitutivity of (open) similarity. Since the latter is defined via the Hybrid primitive notion of substitution, namely subst, our mechanization is rather convoluted and uses several Hybrid infra-structural lemmata concerning the interaction between abstractions and substitution.
2. If $[] \vdash \mathsf{Fun}\, x.E\, x \preccurlyeq^\bullet p$, then $p \Downarrow \mathsf{Fun}\, y.F\, y$ where $\mathsf{cloAbstr}\, F$ and for every closed p' we have $[] \vdash E\, p' \preccurlyeq^\bullet F\, p'$. 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.
3. If $[] \vdash p \preccurlyeq^\bullet q$ and $p \Downarrow v$, then $[] \vdash v \preccurlyeq^\bullet q$. 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: the script consists of about 50 lines.

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

Proof. (Of Theorem 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 $\Gamma \vdash e_1 \preccurlyeq^\circ e_2$, we have to prove that the λ-closures of e_1 and e_2 are related

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

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

by closed simulation. However, having proved that the Howe relation is substitutive, this will follow if we can prove that

$$[] \vdash p \preccurlyeq^{\bullet} q \Longrightarrow [] \vdash p \preccurlyeq^{\circ} q$$

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

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 traditional De Bruijn approach.

The only other comparable mechanized proof about bisimulation in the lazy λ-calculus, to the best of our knowledge is [19]. This follows the Weak HOAS [20] 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 coincides with observational equivalence not via Howe's method, but following Stoughton's adaptation of Milner's context lemma reported in [3]. As well known, this approach does not scale well to even slightly more expressive functional languages; moreover, the formalization heavily relies on *vectors* of terms, to encode a linearized notion of bisimulation, viz. $M_1 \preccurlyeq M_2$ iff for every vector $\boldsymbol{N}$ if $M_1\, \boldsymbol{N} \Downarrow$ then $M_2\, \boldsymbol{N} \Downarrow$. Although conceptually non-problematic, several steps have not been fully verified yet (Marino Miculan, personal communication).

This approach has been more successful in the context of the π-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 β-conversion can implement object-level substitution, which in this case is simply "name" for bound variable in a process.

In [18] the authors presents an encoding of CCS in $FO\lambda^{\Delta\mathbb{N}}$. 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* [21].

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 β expansion, which, in the notation of this paper, would read

$$[\![\ \mathsf{cloExp}\ r;\ \mathsf{cloExp}\ s;\ \forall t.r \Downarrow t \longrightarrow$$
$$(\exists u.s \Downarrow u \wedge (\forall p.\ \mathsf{cloExp}\ p \rightarrow\ (t\ @\ p) \preccurlyeq\ (u\ @\ p)))\ ]\!] \Longrightarrow\ r \preccurlyeq\ s$$

However, a consequence was a series of lemmas concerning Kleene equivalence which is avoided in the Hybrid approach. 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 λ-calculus. 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. As this is due to the fundamental incompatibility of non-stratified hypothetical judgments and (co)induction, this is intrinsically problematic. Nevertheless, some room for improvement is foreseeable:

- 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 wrt 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 [16] coincides with Lévy-Longo tree equivalence—not applicative bisimulation.
- We could try to formulate the Howe relation on closed term with *stratified* hypothetical judgments akin to the way provability can be encoded by switching from natural deduction to sequent calculus, as discussed for example in [17]. The practicality of this approach needs to be supported experimentally.
- We may choose to embrace the two-level approach of Miller & McDowell [17], 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 $FO\lambda^{\Delta N}$ 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 is to use Hybrid in place of a system such as Coq as the meta-meta-logic for the latter architecture; this would have 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.
- Coinductive principles are immediately available, while $FO\lambda^{\Delta N}$, 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, 17].

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

1. A. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In J.J. Joyce and C.-J.H. Seger, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*, pages 414–427, Vancouver, Canada, Aug. 1993. University of British Columbia, Springer-Verlag, published 1994.
2. S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.
3. S. Abramsky and L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1992.
4. S. Ambler, R. Crole, and A. Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. 2002. Submitted, available at http://www.mcs.le.ac.uk/ amomigliano/isabelle/hybrid2.html.
5. S. J. Ambler and R. L. Crole. Mechanised Operational Semantics via (Co)Induction. In *Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 221–238. Springer-Verlag, 1999.
6. N. Benton and A. Kennedy. Monads, effects and transformations. In *Proceedings of the 3rd International Workshop in Higher Order Operational Techniques in Semantics*, volume 26 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998.
7. I. Cervesato and F. Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, pages 264–275, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.
8. J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 124–138, Edinburgh, Scotland, Apr. 1995. Springer-Verlag LNCS 902.
9. L.-H. Eriksson. Pi: An interactive derivation editor for the calculus of partial inductive definitions. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 821–825, Nancy, France, June 1994. Springer Verlag LNAI 814.

10. A. Felty. Two-level meta-reasoning in Coq. 2002. Submitted.
11. J. Frost. A case study of co-induction in Isabelle. Technical Report 359, University of Cambridge, Computer Laboratory, Feb. 1995. Revised version of CUCL 308, August 1993.
12. M. Hofmann. Semantical analysis for higher-order abstract syntax. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 204–213, Trento, Italy, July 1999. IEEE Computer Society Press.
13. F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. ICALP'01*, number 2076 in LNCS, pages 963–978. Springer-Verlag, 2001.
14. F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theories. *Theoretical Computer Science*, 2(253):239–285, 2001.
15. D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
16. S. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In M. M. Stephen Brookes, Achim Jung and A. Scedrov, editors, *Electronic Notes in Theoretical Computer Science*, volume 20. Elsevier Science Publishers, 2000.
17. R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.
18. R. McDowell, D. Miller, and C. Palamidessi. Encoding transition systems in sequent calculus. *Theoretical Computer Science*, 197(1–2):246–272, May 1998.
19. M. Miculan. Developing (meta)theory of lambda-calculus in the theory of contexts. In S. Ambler, R. Crole, and A. Momigliano, editors, *MERLIN 2001: Proceedings of the Workshop on MEchanized Reasoning about Languages with variable bINding*, volume 58 of *Electronic Notes in Theoretical Computer Scienc*, pages 1–22, November 2001.
20. A. Momigliano, S. Ambler, and R. Crole. A comparison of formalizations of the meta-theory of a language with variable bindings in Isabelle. In *Suppplementary Proceedings of TPHOLs 2001, Edimburgh University Technical Report*, 2001.
21. F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.
22. A. M. Pitts. Operationally based theories of program equivalence. Technical report, Cambridge University Computer Laboratory, 1995. Notes to accompany lectures given at the Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK.
23. C. Schürmann. *Automating the Meta-Theory of Deductive Systems.* PhD thesis, Carnegie-Mellon University, 2000. CMU-CS-00-146.