Hybrid Adequacy

R. L. Crole (R.Crole@mcs.le.ac.uk)

Department of Computer Science,

University of Leicester,

University Road,

Leicester, LE1 7RH, U.K.

ABSTRACT

The Hybrid system, implemented within Isabelle HOL, allows object logics to be represented using higher order abstract syntax (HOAS), and reasoned about using tactical theorem proving in general and principles of (co)induction in particular. The form of HOAS provided by Hybrid is essentially lambda calculus with constants. In this paper, we present a particular logical framework which can be viewed as a model of Hybrid, and state and prove that the model is representationally adequate for the lambda calculus with constants. Of fundamental interest is the form of lambda abstractions provided by Hybrid—each abstraction is actually a definition of a de Bruijn expression—and hence the formal system contains a hybrid of named and nameless bound variable notation. In particular the proof of adequacy involves a number of proofs by induction which involve higher order symbolic computations.

1 Introduction

Many people are concerned with the development of computing systems which can be used to reason about and prove properties of programming languages [1–3,14,16,26]. In previous work [2], we developed the Hybrid logical system, implemented in Isabelle HOL, for exactly this purpose. *In this paper we develop an underpinning theory for Hybrid.* The key features of Hybrid are:

- it provides a form of *logical system* within which the syntax of an object level logic can be adequately represented by *higher order abstract syntax* (HOAS);
- it is consistent with *tactical theorem proving* in general, and principles of *induction and coinduction* in particular; and

- it is *definitional* which guarantees consistency *within a classical type theory.*

We will give an overview of the first feature, which should provide sufficient background for our results; the other features have been discussed in [2], [3]. We can then proceed to prove our main theorem, *an adequacy result for the Hybrid system.* Please note that this result was mentioned in [2] without any proofs; this is the first complete account. Informally, this result shows that Hybrid does yield a form of HOAS, into which object level logics can be translated and reasoned about. More formally, regarding HOAS as a λ-calculus with constants, we will prove that there is a representationally adequate mapping (see [11,21]) from the usual mathematical system of λ-calculus into an idealised mathematical model of Hybrid.

We proceed as follows. In Section 2 we explain the essence of the variable binding mechanizm of Hybrid. In Section 3 we prove in detail that de Bruijn notation provides a representationally adequate model of the λ-calculus. In order to achieve this, we make use of the folklore result that (proper) de Bruijn expressions and λ-calculus expressions are in bijective correspondence; in fact this is proved in the Appendix A—we include a proof as doing so allows us to set up our own notation which is crucial for giving intensional details within our proofs of Hybrid adequacy. In Section 4 we state and prove Hybrid adequacy, our central result, appealing to Section 3. We conclude in Section 5.

2 An Overview of Hybrid

2.1 A Novel Variable Binding Mechanizm

In developing Hybrid, our goal was to define a datatype for λ-calculus with constants over which we can deploy (co)induction principles. This datatype was thought of as a form of HOAS in which variable binding was to be realized by Isabelle HOL's internal meta-variable binding. The goal of this section is to give a high level overview of the development of the key ideas, which were originally developed in collaboration with Simon Ambler and Alberto Momigliano.

The starting point was the work [4] of Andrew Gordon, which we briefly review. It is well known (though rarely proved—see Section 3 and Appendix A) that λ-calculus expressions are in bijection with (a subset of) de Bruijn expressions. Gordon defines a notation in which expressions have *named free variables* given by *strings.* He can write $E = \mathsf{dLAMBDA}\,\mathtt{v}\,e$ (where $\mathtt{v}$ is a string) which corresponds to a λ-abstraction in which $\mathsf{dLAMBDA}$ acts as the usual λ-calculus binder. But in fact $\mathsf{dLAMBDA}$ is a higher order function. The

function dLAMBDA is defined so that E is actually equal to a de Bruijn expression which has an outer abstraction, and an immediate subexpression which is e in de Bruijn form (in which any of the free occurrences of v in e—which were bound by the outer binder dLAMBDA in E—have been converted to bound de Bruijn indices). For example,

$$\mathsf{dLAMBDA}\,\mathtt{v}\,(\mathsf{dAPP}\,(\mathsf{dVAR}\,\mathtt{v})\,(\mathsf{dVAR}\,\mathtt{u})) = \mathsf{dABS}\,(\mathsf{dAPP}\,(\mathsf{dBND}\,0)\,(\mathsf{dVAR}\,\mathtt{u}))$$

Gordon demonstrates the utility of this approach within a theorem prover. The user may work with expressions which have explicit named binding variables (such as the v above), but such expressions are automatically converted to de Bruijn form by the machine, with which it is easier for the machine to work. The approach provides a good mechanizm through which the user may work with named bound variables, but it does not exploit the built in meta-level α-equivalence which a theorem prover typically possesses. What would be ideal is a system in which the string binders of Gordon are replaced by meta-level binders. This is exactly what Hybrid achieves, and is one of the key novelties of our approach.

Hybrid provides a binding mechanizm similar to dLAMBDA. Gordon's E would be written as $\mathsf{LAM}\,v.\,e$ in Hybrid. This is *also* a *definition* for a de Bruijn expression; and $\mathsf{LAM}\,v.\,e$ can indeed be proved equal to an actual de Bruijn expression. A *crucial difference* in our approach is that *bound variables* are actually *bound meta-variables in Isabelle HOL*. Thus the v in $\mathsf{LAM}\,v.\,e$ is a meta-variable (and not a string as in Gordon's approach) allowing us to exploit the meta-level α-equivalence.

We give some illustrative examples. Of central importance in the actual implementation of Hybrid is a datatype of de Bruijn expressions, where i and j are de Bruijn indices, and ν are names for constants:

$$E ::= \mathsf{CON}\ \nu \mid \mathsf{VAR}\ i \mid \mathsf{BND}\ j \mid E\ \$\$\ E \mid \mathsf{ABS}\ E$$

Let $E_O = \lambda\, v_8.\, \lambda\, v_2.\, v_8\ v_3$. Gordon would represent this by

$$E_G = \mathsf{dLAMBDA}\,\mathtt{v8}\,(\mathsf{dLAMBDA}\,\mathtt{v2}\,(\mathsf{dAPP}\,(\mathsf{dVAR}\,\mathtt{v8})\,(\mathsf{dVAR}\,\mathtt{v3})))$$

which equals dABS (dABS (dAPP (dBND 1) (dVAR v3))). In Hybrid we choose to denote object level free variables by expressions of the form VAR i. This has essentially no impact on the key technical details relating to binders. In Hybrid the E_O above is rendered as $E_H = \mathsf{LAM}\,v_8.\,(\mathsf{LAM}\,v_2.\,(v_8\ \$\$\ \mathsf{VAR}\ 3))$. In Hybrid E_H is equal to ABS (ABS (BND 1 $$ VAR 3)), with the overall effect

being analogous to Gordon's approach, but the underlying definitions and details being very different.

To summarise,

- object level free variables v_i are defined as Hybrid expressions of the form $\mathsf{VAR}\ i$;
- object level bound variables v_j are defined as Hybrid (bound) meta-variables v_j;
- object level abstractions $\lambda\, v_j.\, E$ are defined as Hybrid expressions $\mathsf{LAM}\, v_j.\, e$; and
- object level applications $E_1\ E_2$ are defined as Hybrid expressions $e_1\ \$\$\ e_2$.

A central aim of this paper is to show that Hybrid really does provide a good representation of the λ-calculus. One can view the summary above as informally specifying a function Π from an object level λ-calculus to Hybrid where (for example) $\Pi\,(\lambda\, v_j.\, E) \stackrel{\text{def}}{=} \mathsf{LAM}\, v_j.\, (\Pi\, E)$. Thus our central theorem is a proof that the function Π, which we shall formally define, is representationally adequate.

2.2 Abstraction and Application Notation

In this paper there are a variety of binding operations for variants of functional abstractions, together with associated applications. This is potentially confusing so we give a look-up table in Figure 1.

We use a standard logical framework to model Hybrid, and the meta-binder is written as Λ. In fact in the Hybrid system, ABS is a Isabelle HOL datatype constructor and Λ is a Isabelle HOL binder. However the details are not of direct concern to this paper which only addresses meta-properties of Hybrid.

3 The Adequacy of de Bruijn Syntax for Lambda Syntax

In this section we prove in a very precise way that de Bruijn expressions provide an *adequate* representation of the expressions of the λ-calculus. Experienced readers may wonder why we are doing this; the result is folklore, and known and used by all. There are two main reasons. Firstly, if one is *really* strict about details, and does not hand-wave, then the result is surprisingly tricky to prove. Secondly, and of much greater importance, we will employ both the result itself, and also much of its proof given in the Appendix, in establishing Hybrid adequacy.

Syntax	Informal Description	Defined on Page
$\mathsf{abs}(D)$	object level de Briujn abstraction	5
$D_1 \$ D_2$	object level de Briujn application	5
$\lambda\, v_i.\, E$	object level lambda abstraction	6
$E_1\ E_2$	object level lambda application	6
$\mathsf{ABS}\ c$	Hybrid de Bruijn abstraction	12
$\mathsf{LAM}\ v_i.\, c$	Hybrid lambda abstraction	16
$c_1 \$\$ c_2$	Hybrid de Briujn application	12
$\Lambda\, v_i.\, c$	logical framework meta-abstraction	13
$c_1\ c_2$	logical framework meta-application	13

Fig. 1. Abstraction Notation

In Section 3.1 we set up our own syntax. In Section 3.2 we state that there is a bijection between the expressions of the λ-calculus and a certain subset of the de Bruijn expressions; note that the proof appears in Appendix A. In Section 3.3 we state and prove this preliminary adequacy result (later used to show Hybrid adequacy).

3.1 de Bruijn Syntax and Lambda Syntax

We assume familiarity with de Bruijn expressions but summarise our own notation. We inductively define a set of de Bruijn expressions. The set of expressions is denoted by $\mathcal{DB}$, with expressions generated by

$$D ::= \mathsf{con}(\nu) \mid \mathsf{var}(i) \mid \mathsf{bnd}(j) \mid \mathsf{abs}(D) \mid D_1 \$ D_2$$

where i and j range over the natural numbers $\mathbb{N}$, and ν over a set of names. One should think of a de Bruijn expression as a finite rooted syntax tree. The leaf nodes are labelled either by constants $\mathsf{con}(\nu)$; by $\mathsf{var}(i)$ which corresponds to a free variable in the λ-calculus; or by $\mathsf{bnd}(j)$ which corresponds to a bound variable in the λ-calculus. We shall employ informal notation for occurrences of subtrees. For example we may write $\mathsf{var}(i) \in D$ or possibly even $i \in D$. We call the j in expressions $\mathsf{bnd}(j)$ **bound indices**. We call the i in expressions $\mathsf{var}(i)$ **free indices**. Given a de Bruijn expression D, a bound index j which occurs in D is said to be **dangling** if j or less abs nodes occur on the path between the index j and the root of D. Otherwize it is not dangling. D is said

to be at **level** l, where $l \geq 0$, if enclosing[1] D inside l nodes, each labelled with abs, ensures that the resulting expression has no dangling indices. We can define a predicate $\mathsf{level}\, n : \mathcal{DB} \to \mathbb{B}$ for each $n \in \mathbb{N}$ where $\mathbb{B} \stackrel{\text{def}}{=} \{T, F\}$ such that the Boolean $\mathsf{level}\, l\, D$ is true just in case D is of level l, by setting

$$\begin{aligned}
\mathsf{level}\, l\, \mathsf{con}(\nu) &= T \\
\mathsf{level}\, l\, \mathsf{var}(i) &= T \\
\mathsf{level}\, l\, \mathsf{bnd}(j) &= l > j \\
\mathsf{level}\, l\, (D_1\ \$\ D_2) &= (\mathsf{level}\, l\, D_1) \wedge (\mathsf{level}\, l\, D_2) \\
\mathsf{level}\, l\, \mathsf{abs}(D) &= \mathsf{level}\, (l+1)\, D
\end{aligned}$$

It is (informally) clear that for any D, a (unique) minimum level m exists, and that D is at level l for any $l \geq m$. There is of course a function $\mathsf{minl} : \mathcal{DB} \to \mathbb{N}$, but we only use it indirectly and leave its definition as a simple exercise.

Let $\mathcal{DB}(l)$ be the set of de Bruijn expressions at level l. We make use of a predicate $\mathsf{proper} : \mathcal{DB} \to \mathbb{B}$ which is by definition $\mathsf{level}\, 0$. Let $\mathcal{PDB} \stackrel{\text{def}}{=} \mathcal{DB}(0)$ be the set of **proper** de Bruijn expressions. A proper expression is one that has no dangling indices (this follows from the formal definition) and (thus) corresponds to a λ-calculus expression. We assume readers are informally familiar with the correspondence—however, we shortly describe it in detail. It follows from the discussion above that

$$\mathcal{PDB} = \mathcal{DB}(0) \subset \mathcal{DB}(1) \subset \ldots \subset \mathcal{DB}(l) \subset \ldots$$

and it is easy to see that $\mathcal{DB} = \bigcup_{l<\omega} \mathcal{DB}(l)$ by considering minimum levels.

We set up a notation for the traditional λ-calculus. The expressions will consist of constants, variables, applications and abstractions. More precisely, we have a countable set of variables, with a typical variable denoted by v_k where $k \geq 0$, that is $k \in \mathbb{N}$. The expressions are inductively defined by the grammar $E ::= \nu \mid v_k \mid \lambda\, v_k.\, E \mid E\ E$. We adopt the usual notions of **free** and **bound** variables, and α-equivalence. For completeness we outline our notation. If v_k occurs in E then we write $v_k \in E$; we omit the usual definition of **occurs in**. We write $fv(E)$ for the set of free variables occuring in E. In abstractions of the form $\lambda\, v_k.\, E$, we refer to the occurrence of v_k immediately after the binder λ as a **binding** occurrence, and the free occurrences of v_k in E are **bound** in $\lambda\, v_k.\, E$. We sometimes call E the **scope** of the abstraction, and in general any variable $v_{k'}$ occurring in any E' is **bound** if it occurs in a sub-expression either as a binding occurrence, or within the scope of a binder. Given expressions E and E', and a variable v_k, then we write $E[E'/v_k]$ for a *unique*† expression

[1] D enclosed by two such nodes is $\mathsf{abs}(\mathsf{abs}(D))$.

which, informally, is E with free occurrences of v_k replaced by E', with renaming to avoid capture. Our definition appears on page 32, and it ensures that the action $(E, E', v_k) \mapsto E[E'/v_k]$ really is a function[†]. We say that v_w is **fresh** for E if the variable has *no occurrences* in the expression. *Having defined* substitution, we can then define α-equivalence. We write $\mathcal{LE}$ for the set of λ-calculus expressions. If expressions E and E' are α-equivalent, we write $E \sim_\alpha E'$. In this paper, α-equivalence is an inductively defined subset of $\mathcal{LE} \times \mathcal{LE}$ generated by formal axioms and rules. There is a single axiom of the form $\lambda\, v_k.\, E \sim_\alpha \lambda\, v_{k'}.\, E[v_{k'}/v_k]$ where $k \neq k'$ and $v_{k'}$ is any variable for which $v_{k'} \notin fv(E)$; structural congruence rules for application and abstraction; plus the usual rules for equivalence relations. In order to define α-equivalence, we must already have defined substitution for λ-calculus. This is treated in great detail in the Appendix, beginning page 31. We write $[E]_\alpha$ for the alpha equivalence class of E and $\mathcal{LE}/\!\sim_\alpha$ for the set of alpha equivalence classes of λ-calculus expressions.

3.2 A Bijection between de Bruijn Expressions and Lambda Expressions

Later on in Section 3, we shall make use of the following theorem.

Theorem 3.1 There is a bijection between the set $\mathcal{LE}/\!\sim_\alpha$ of alpha equivalence classes of λ-calculus expressions, and the set $\mathcal{PDB}$ of proper de Bruijn expressions,

$$\theta : \mathcal{LE}/\!\sim_\alpha \rightleftharpoons \mathcal{PDB} : \phi$$

The proof of this result is relegated to Appendix A. However, since in Section 4 we are going to make use of some of the notation and definitions which are used within the proof, we state these here and now.

A **list** L is one whose elements (if any) are object level variables v_k. We write ϵ for the **empty list**, and write v_k, L and L, L' for **cons** and **concatenation**. Thus a typical non-empty list is $v_{10}, v_{70}, v_{70}, v_6, v_2, v_0, v_{10}$. If a list L is non-empty, the head has **position** 0, and the last element **position** $|L| - 1$ where $|L|$ is the **length** of the list. Thus v_{70} occurs at positions 1 and 2 in the recent example. If L is non-empty and v_k occurs in it, we write $v_k \in L$. Suppose also that the *first* occurrence is at position p. Then we write $\mathsf{pos}\, v_k\, L$ for p. If $v_k \notin L$ then $\mathsf{pos}\, v_k\, L$ is undefined. We write $\mathsf{el}\, p\, L$ for the variable v_k at position p if such exists; otherwise $\mathsf{el}\, p\, L$ is undefined. We say that L is **ordered** if L is a list, has no repeated elements, and the indices occur in decreasing order. Thus a typical non-empty ordered list is $v_{100}, v_7, v_6, v_2, v_0$. If S is a set of variables, we will use informal notation such as $S \cap L$ to mean the intersection of S and the set of variables in L.

In order to prove the theorem, we will establish the existence of a certain family of pairs of functions

$$[\![-]\!]_L : \mathcal{LE} \rightleftarrows \mathcal{DB}(|L|) : (\!|-|\!)_L$$

satisfying $\theta \circ q = [\![-]\!]_\epsilon$ and $\phi = q \circ (\!|-|\!)_\epsilon$ with q the quotient function for α-equivalence. We will make use of these functions here and also in Section 4, so we assert their existence, but leave the proofs for later on.

Proposition 3.2 For any L, there is a function $[\![-]\!]_L : \mathcal{LE} \to \mathcal{DB}(|L|)$ given recursively by the clauses below; in particular, $[\![-]\!]_\epsilon : \mathcal{LE} \to \mathcal{PDB}$.

- $[\![\nu]\!]_L \stackrel{\text{def}}{=} \mathsf{con}(\nu)$
- On variables we define

$$[\![v_i]\!]_L \stackrel{\text{def}}{=} \begin{cases} \mathsf{bnd}(\mathsf{pos}\, v_i\, L) \text{ if } v_i \in L \\ \mathsf{var}(i) \text{ if } v_i \notin L \end{cases}$$

- $[\![E_1\ E_2]\!]_L \stackrel{\text{def}}{=} [\![E_1]\!]_L\ \$\ [\![E_2]\!]_L$
- $[\![\lambda\, v_i.\, E]\!]_L \stackrel{\text{def}}{=} \mathsf{abs}([\![E]\!]_{v_i, L})$

Proposition 3.3 For any ordered L, there is a function $(\!|-|\!)_L : \mathcal{DB}(|L|) \to \mathcal{LE}$ given recursively by the clauses below; in particular, $(\!|-|\!)_\epsilon : \mathcal{PDB} \to \mathcal{LE}$.

- $(\!|\mathsf{con}(\nu)|\!)_L \stackrel{\text{def}}{=} \nu$
- $(\!|\mathsf{var}(i)|\!)_L \stackrel{\text{def}}{=} v_i$
- $(\!|\mathsf{bnd}(j)|\!)_L \stackrel{\text{def}}{=} \mathsf{el}\ j\ L$
- $(\!|D_1\ \$\ D_2|\!)_L \stackrel{\text{def}}{=} (\!|D_1|\!)_L\ (\!|D_2|\!)_L$
- $(\!|\mathsf{abs}(D)|\!)_L \stackrel{\text{def}}{=} \lambda\, v_{M+1}.\, (\!|D|\!)_{v_{M+1}, L}$ where $M \stackrel{\text{def}}{=} \mathsf{Max}(D; L)$ with

$$\mathsf{Max}(D; L) \stackrel{\text{def}}{=} \mathsf{Max}\ \{i \mid \mathsf{var}(i) \in D\} \bigcup \underbrace{\{j \mid \mathsf{head}(L) = v_j\}}_{\varnothing \text{ if } L \text{ empty}}$$

We take $\mathsf{Max}\ \varnothing \stackrel{\text{def}}{=} 0$. Informally $\mathsf{Max}(D; L)$ denotes the maximum of the free indices which occur in D and the indices of L.

3.3 Adequacy of de Bruijn for the λ-calculus

Before proving an adequacy result for Hybrid, we prove that de Bruijn expressions give rise to an adequate representation of the λ-calculus; this is

Theorem 3.8. Before stating and proving our result formally, we give some lemmas and propositions. We begin by showing in Lemma 3.4 that our notion of substitution for de Bruijn expressions (cite??) is well-defined. We then prove Lemma 3.5 and Lemma 3.6 which will be used to prove Proposition 3.7 which shows that the functions $[\![-]\!]_L$ commute with substitution. We can then prove Theorem 3.8.

Lemma 3.4 For any $m \geq m' \geq 0$ and $k \geq 0$ there is a function $\mathcal{DB}(m) \times \mathcal{DB}(m') \times \mathbb{N} \to \mathcal{DB}(m)$ given by $(D, D', k) \mapsto D[D'/\mathsf{var}(k)]$, which, informally, maps (D, D', k) to the expression D in which all occurrences of $\mathsf{var}(k)$ are replaced by D'.

Proof The substitution function can be defined recursively in the expected way. Note that there is of course no notion of variable renaming—thus the definition is straightforward and omitted. One does need to prove that the function has the stated source and target but we omit the easy proof. □

Lemma 3.5 For any $E \in \mathcal{LE}$, list L and variable v_k, if either $v_k \notin fv(E)$ or $v_k \in L$, then $\mathsf{var}(k) \notin [\![E]\!]_L$.

Proof The proof is by a simple induction over $E \in \mathcal{LE}$. □

Lemma 3.6 Let $E \in \mathcal{LE}$, let L and L' be any lists, and v_i a variable. Further, suppose that for any $v_k \in fv(E)$, either $v_k \in L'$ or $v_k \notin v_i, L$. Then $[\![E]\!]_{L',v_i,L} = [\![E]\!]_{L',L}$.

Proof The proof is by a simple induction over $E \in \mathcal{LE}$. □

Proposition 3.7 For any expressions $E, E' \in \mathcal{LE}$, list L, and variable v_k, if $v_k \notin L$ and $fv(E') \cap L = \varnothing$, then

$$[\![E[E'/v_k]]\!]_L = [\![E]\!]_L[[\![E']\!]_L/\mathsf{var}(k)] \qquad (*)$$

Proof The substitution functions exist by appeal to Lemma 3.4 and the Appendix. The proof proceeds by induction over the size of the expression E—we adopt the size notation from the proof of Lemma A.12 on page 39. Write $\mathsf{size}(E)$ for the size of E and $\Phi(E)$ for $(*)$ in which E', L, and k are universally quantified and satisfy the given constraints. Write $\Psi(n)$ for $(\forall E)(\mathsf{size}(E) = n \Longrightarrow \Phi(E))$ and we prove $\forall n.\Psi(n)$ by strong induction on n. We write *LHS* and *RHS* for the appropriate instance of $(*)$ in the inductive steps below.

$\boxed{\Psi(1)}$ If E is a constant the result is immediate. Else choose E to be v_i, of size 1, and prove $\Phi(v_i)$.

(Case $i = k$):

$$LHS = [\![E']\!]_L = \mathsf{var}(i)[[\![E']\!]_L/\mathsf{var}(k)] = [\![v_i]\!]_L[[\![E']\!]_L/\mathsf{var}(k)] = RHS$$

with the third equality following because $v_i = v_k \notin L$.

(Case $i \neq k$):

$$LHS = [\![v_i]\!]_L = [\![v_i]\!]_L[[\![E']\!]_L/\mathsf{var}(k)] = RHS$$

where if $v_i \in L$ the second equality is immediate; and if not, the equality follows because $i \neq k$.

$\boxed{(\forall n)[(\forall m < n)(\Psi(m)) \Longrightarrow \Psi(n)]}$ where $n \geq 2$. Consider the case when the expression is $\lambda\, v_i.\, E$ and $\mathsf{size}(\lambda\, v_i.\, E) = n$. We prove $\Phi(\lambda\, v_i.\, E)$.

(Case $i = k$):

$$LHS = [\![\lambda\, v_i.\, E]\!]_L = \mathsf{abs}([\![E]\!]_{v_i,L}) =$$
$$\mathsf{abs}([\![E]\!]_{v_i,L}[[\![E']\!]_L/\mathsf{var}(k)]) = \mathsf{abs}([\![E]\!]_{v_i,L})[[\![E']\!]_L/\mathsf{var}(k)] = RHS$$

where the third equality follows from Lemma 3.5 since $v_k = v_i \in v_i, L$ and so $\mathsf{var}(k) = \mathsf{var}(i) \notin [\![E]\!]_{v_i,L}$.

(Case $i \neq k$): We examine sub-cases according to whether the substitution involves a name clash or not.

(Subcase $v_k \notin fv(\lambda\, v_i.\, E)$ or $v_i \notin fv(E')$): If $v_k \notin fv(\lambda\, v_i.\, E)$ we have

$$LHS = [\![\lambda\, v_i.\, E]\!]_L = \mathsf{abs}([\![E]\!]_{v_i,L}) =$$
$$\mathsf{abs}([\![E]\!]_{v_i,L}[[\![E']\!]_L/\mathsf{var}(k)]) = \mathsf{abs}([\![E]\!]_{v_i,L})[[\![E']\!]_L/\mathsf{var}(k)] = RHS$$

where the third equality follows from Lemma 3.5 because $v_k \notin fv(\lambda\, v_i.\, E)$ and $i \neq k$ imply $v_k \notin fv(E)$.

If $v_i \notin fv(E')$

$$\begin{aligned} LHS &= [\![\lambda\, v_i.\, E[E'/v_k]]\!]_L && (1)\\ &= \mathsf{abs}([\![E[E'/v_k]]\!]_{v_i,L}) && (2)\\ &= \mathsf{abs}([\![E]\!]_{v_i,L}[[\![E']\!]_{v_i,L}/\mathsf{var}(k)]) && (3) \end{aligned}$$

$$= \mathsf{abs}(\llbracket E \rrbracket_{v_i,L}[\llbracket E' \rrbracket_L / \mathsf{var}(k)]) \tag{4}$$
$$= RHS \tag{5}$$

where equality (3) follows by induction as $\mathsf{size}(E) = n - 1$ and so $\Phi(E)$ holds, and $v_k \notin v_i, L$ and $fv(E') \cap (v_i, L) = \varnothing$; and equality (4) follows from an instance of Lemma 3.6 in which $L' = \epsilon$ and again $fv(E') \cap (v_i, L) = \varnothing$.

(Subcase $v_k \in fv(\lambda\, v_i.\, E)$ and $v_i \in fv(E')$):

$$LHS = \llbracket \lambda\, v_w.\, E[v_w/v_i][E'/v_k] \rrbracket_L \tag{6}$$
$$= \mathsf{abs}(\llbracket E[v_w/v_i][E'/v_k] \rrbracket_{v_w,L}) \tag{7}$$
$$= \mathsf{abs}(\llbracket E[v_w/v_i] \rrbracket_{v_w,L}[\llbracket E' \rrbracket_{v_w,L} / \mathsf{var}(k)]) \tag{8}$$
$$= \mathsf{abs}(\llbracket E \rrbracket_{v_i,L}[\llbracket E' \rrbracket_L / \mathsf{var}(k)]) \tag{9}$$
$$= RHS \tag{10}$$

where equality (8) follows by induction since

$$\mathsf{size}(E[v_w/v_i]) = \mathsf{size}(E) = n - 1$$

and further w is the maximum of the indices in E, E' and v_k, plus 1; and equality (9) follows by appeal to Lemma A.12 and Lemma 3.6. The details for applications are easy and omitted. □

Theorem 3.8 [Representational Adequacy] The function

$$\theta : \mathcal{LE}/\!\sim_\alpha \to \mathcal{PDB} \subset \mathcal{DB}$$

is **representationally adequate** in the sense that

Bijective it is bijective; and
Commutes with Substitution which means that

$$\theta([E]_\alpha[[E']_\alpha/v_k]) = \theta([E]_\alpha)[\theta([E']_\alpha)/\mathsf{var}(k)]$$

Proof

Bijective is immediate from Theorem 3.1.
Commutes with Substitution is immediate from Proposition 3.7 and the definition of θ on page 42.

□

$$\nu :: con$$

$$i :: var$$

$$j :: bnd$$

$$\mathsf{CON} :: con \Rightarrow exp$$

$$\mathsf{VAR} :: var \Rightarrow exp$$

$$\mathsf{BND} :: bnd \Rightarrow exp$$

$$\$\$ :: exp \Rightarrow exp \Rightarrow exp$$

$$\mathsf{ABS} :: exp \Rightarrow exp$$

Fig. 2. Constructor Constants

4 An Adequacy Result for Hybrid

4.1 A Model of Hybrid

The specific goal of this section is to describe a mathematical model of Hybrid and then show that the model provides an adequate representation of the λ-calculus. Our informal motivation is to provide a mathematical underpinning for Hybrid which renders clearly its key principles as mathematical theorems. Thus we hope that this paper will be of special benefit to those who might wish to use Hybrid. Please recall the informal description of Π at the end of Section 2.1. Our central result is the adequacy Theorem 4.3 which states that there is a particular function $\Pi_\epsilon : \mathcal{LE}/\!\sim_\alpha \rightarrow \mathcal{CLF}_\epsilon(exp)$ which is bijective (onto its image) and compositional on substitution, where $\mathcal{CLF}_\epsilon(exp)$ is our model of Hybrid shortly to be defined.

It would very tedious to provide a "complete" proof of the adequacy of the fully implemented Isabelle HOL Hybrid tool for λ-calculus. We could work with a formal system for higher order logic, but that would lead to long and complex proofs. Here we take a simpler approach—*we present Hybrid syntax as a theory in a logical framework*. For logical frameworks, see, for example [11,21] and perhaps [1,12]. The meta-variables of the framework play the role of the actual Isabelle HOL meta-variables of implemented Hybrid. The theory has ground types *con*, *var*, *bnd*, and *exp*, ranged over by γ. The (higher) types are given by $\sigma ::= \gamma \mid \sigma \Rightarrow \sigma$. We declare constructor constants in Figure 2 where i and j range over the natural numbers, and ν over a set of names for (object level) constants. We shall use κ to range over the constructor constants of the theory. The judgements are generated using the standard type assignment system of such a logical framework. More precisely, suppose that Γ is a **context**, that

$$\frac{\Gamma(v_k) = \sigma_1 \Rightarrow \sigma_2 \Rightarrow \ldots \sigma_n \Rightarrow \gamma \qquad \Gamma \vdash_{can} c_i :: \sigma_i \quad (0 \leq i \leq n)}{\Gamma \vdash_{can} v_k\, \vec{c} :: \gamma}\ \text{VAR}$$

$$\frac{\kappa :: \sigma_1 \Rightarrow \sigma_2 \Rightarrow \ldots \sigma_n \Rightarrow \gamma \qquad \Gamma \vdash_{can} c_i :: \sigma_i \quad (0 \leq i \leq n)}{\Gamma \vdash_{can} \kappa\, \vec{c} :: \gamma}\ \text{CST}$$

$$\frac{\Gamma, v_k :: \sigma \vdash_{can} c :: \sigma'}{\Gamma \vdash_{can} \Lambda\, v_k.\, c :: \sigma \Rightarrow \sigma'}\ \text{ABS}$$

Fig. 3. Inductive Definition of Canonical Forms

is, finite partial function from the set of framework meta-variables to types. Then the type assignment system has judgements of the form $\Gamma \vdash e :: \sigma$. We omit the (usual) inductive definition. We define $\mathcal{LF}_\sigma(\Gamma) \stackrel{\text{def}}{=} \{e \mid \Gamma \vdash e :: \sigma\}$ and $\mathcal{LF} \stackrel{\text{def}}{=} \bigcup_{\sigma,\Gamma} \mathcal{LF}_\sigma(\Gamma)$. We give the inductive definition of canonical forms c in Figure 3. They are introduced using the judgements $\Gamma \vdash_{can} c :: \sigma$. We write $\mathcal{CLF}_\sigma(\Gamma) \stackrel{\text{def}}{=} \{c \mid \Gamma \vdash_{can} c :: \sigma\}$. Given a list of meta-variables $L = v_{k_1}, \ldots, v_{k_m}$, then we write Γ^L_{exp} for the context (partial function) $v_{k_1} :: exp, \ldots, v_{k_m} :: exp$. Note that it is standard to prove that $\mathcal{CLF}_\sigma(\Gamma) \subset \mathcal{LF}_\sigma(\Gamma)$.

To formally state the adequacy theorem we need an auxiliary function lbnd, whose existence is stated in the following proposition. We prove the proposition, and a lemma that the function respects a simple form of α-equivalence, before stating the theorem. After the theorem, we will have sufficient notation set up to give an informal explanation of lbnd; however, readers may care to skip ahead to Theorem 4.3 and explanatory Remark 4.4 before returning here.

Proposition 4.1 For all $n \geq 0$ and lists L there is a function with the following source and target

$$\mathsf{lbnd}\ n : \mathcal{CLF}_{exp \Rightarrow exp}(\Gamma^L_{exp}) \to \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

with a recursive definition as follows

- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{CON}\ \nu) \stackrel{\text{def}}{=} \mathsf{CON}\ \nu$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, v_k) \stackrel{\text{def}}{=} \mathsf{BND}\ n$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, v_{k'}) \stackrel{\text{def}}{=} v_{k'}$ where $k \neq k'$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{VAR}\ i) \stackrel{\text{def}}{=} \mathsf{VAR}\ i$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{BND}\ j) \stackrel{\text{def}}{=} \mathsf{BND}\ j$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c_1\ \$\$\ c_2) \stackrel{\text{def}}{=} (\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c_1))\ \$\$\ (\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c_2))$
- $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{ABS}\ c) \stackrel{\text{def}}{=} \mathsf{ABS}\ (\mathsf{lbnd}\ (n+1)\ (\Lambda\, v_k.\, c))$

Proof The proof is subtle, and we appeal to strong induction on heights of derivations. Let

$$\Phi(\Gamma \vdash_{can} c :: \sigma) \stackrel{\text{def}}{=} (\forall n \geq 0)(\forall L)$$
$$(\Gamma = \Gamma^L_{exp} \wedge \sigma = exp \Rightarrow exp \Longrightarrow \mathsf{lbnd}\ n\ c \in \mathcal{CLF}_{exp}(\Gamma^L_{exp}))$$

and

$$\Psi(h) \stackrel{\text{def}}{=} (\forall \Gamma \vdash_{can} c :: \sigma)(\mathsf{hgt}(\Gamma \vdash_{can} c :: \sigma) = h \Longrightarrow \Phi(\Gamma \vdash_{can} c :: \sigma))$$

We prove $\forall h \in \mathbb{N}.\Psi(h)$ by strong induction.

$\boxed{\Psi(0)}$ The only possible derivations of height 0 come from VAR and CST. These cases are vacuous, as no ground type γ is the higher type $exp \Rightarrow exp$.

$\boxed{(\forall h)[(\forall h' < h)(\Psi(h')) \Longrightarrow \Psi(h)]}$ Pick arbitrary $h \geq 1$ and derivation with $\mathsf{hgt}(\Gamma \vdash_{can} \hat{c} :: \sigma) = h$. We have to prove $\Phi(\Gamma \vdash_{can} \hat{c} :: \sigma)$. If the final rule in the derivation comes from VAR or CST, we are done as no ground type γ is the higher type $exp \Rightarrow exp$.

Hence the final rule is ABS. So suppose that in fact $\Gamma^L_{exp}, v_k :: exp \vdash_{can} c :: exp$ where $\hat{c} \equiv \Lambda\, v_k.\, c$. We need to show that $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c) \in \mathcal{CLF}_{exp}(\Gamma^L_{exp})$.

We must now consider all the possible ways this judgement could have been derived. It cannot have been derived using ABS, as exp is not a higher type.

If it was derived using VAR, then for some list $\vec{c'}$ of canonical forms we must have $c \equiv v_{k'}\, \vec{c'}$. But $(\Gamma^L_{exp}, v_k :: exp)v_{k'} = exp$, a ground type, and so the list must be empty and $c \equiv v_{k'}$. Then $\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c) \in \mathcal{CLF}_{exp}(\Gamma^L_{exp})$ is trivial by appeal to the definition.

If it was derived using CST, then for some list $\vec{c'}$ of canonical forms we must have $c \equiv \kappa\, \vec{c'}$. But c has type exp, and so κ can only be CON, VAR, BND, \$\$ or ABS. We give the remaining details for ABS only; the other cases are similar. So, for some c' we have $\Gamma^L_{exp}, v_k :: exp \vdash_{can} \mathsf{ABS}\ c' :: exp$ and so $\Gamma^L_{exp}, v_k :: exp \vdash_{can} c' :: exp$. But this has a derivation height

$$\mathsf{hgt}(\Gamma^L_{exp} \vdash_{can} \Lambda\, v_k.\, c :: exp \Rightarrow exp) - 2 = h - 2$$

and so

$$\mathsf{hgt}(\Gamma^L_{exp} \vdash_{can} \Lambda\, v_k.\, c' :: exp \Rightarrow exp) = h - 1$$

Thus we can apply the (strong) induction hypothesis

$$\Phi(\Gamma^L_{exp} \vdash_{can} \Lambda\, v_k.\, c' :: exp \Rightarrow exp)$$

to deduce that $\mathsf{lbnd}\ (n+1)\ (\Lambda\, v_k.\, c') \in \mathcal{CLF}_{exp}(\Gamma^L_{exp})$. But then

$$\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{ABS}\ c') = \mathsf{ABS}\ (\mathsf{lbnd}\ (n+1)\ (\Lambda\, v_k.\, c')) \in \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

as required. □

Lemma 4.2 Whenever $\Gamma^L_{exp} \vdash_{can} \Lambda\, v_k.\, c :: exp \Rightarrow exp$, we have

$$\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, c) = \mathsf{lbnd}\ n\ (\Lambda\, v_{k'}.\, c[v_{k'}/v_k])$$

provided that $v_{k'}$ does not occur free in c.

Proof Both the formal statement and stages of the proof are very similar to those of Proposition 4.1. We give just a very few informal details.

Suppose that the final rule used in the derivation is ABS. Of course

$$\Gamma^L_{exp}, v_k :: exp \vdash_{can} c :: exp \qquad (*)$$

We must now consider all the possible ways this judgement could have been derived. It cannot have been derived using ABS, as exp is not a higher type.

If $(*)$ was derived using VAR, reasoning just as in Proposition 4.1 we get $c \equiv v_{k''}$ for some k''. If $k \neq k''$ we have

$$\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, v_{k''}) \stackrel{\text{def}}{=} v_{k''} = \mathsf{lbnd}\ n\ (\Lambda\, v_{k'}.\, v_{k''}) = \mathsf{lbnd}\ n\ (\Lambda\, v_{k'}.\, v_{k''}[v_{k'}/v_k])$$

where the assumption that $v_{k'}$ is not free implies that $k' \neq k''$. In the case $k = k''$ both expressions equal $\mathsf{BND}\ n$.

We omit all details in the case $(*)$ was derived using CST; reason exactly as in Proposition 4.1 and calculate from the definitions. □

Let us state our central theorem. Once we have done this we give a short explanation of the lbnd function.

Theorem 4.3 [Representational Adequacy] There is a well-defined function

$$\Pi_\epsilon : \mathcal{LE}/\!\sim_\alpha \to \Pi_\epsilon\,(\mathcal{LE}/\!\sim_\alpha) \subset \mathcal{CLF}_{exp}(\epsilon)$$

arising from the family of well-defined functions

$$\Pi_L : \mathcal{LE}/\!\sim_\alpha \to \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

given recursively by

- $\Pi_L\,([\nu]_\alpha) \stackrel{\text{def}}{=} \mathsf{CON}\ \nu$
- $$\Pi_L\,([v_i]_\alpha) \stackrel{\text{def}}{=} \begin{cases} v_i \text{ if } v_i \in L \\ \mathsf{VAR}\ i \text{ if } v_i \notin L \end{cases}$$
- $\Pi_L\,([E_1\ E_2]_\alpha) \stackrel{\text{def}}{=} (\Pi_L\,[E_1]_\alpha)\ \$\$\ (\Pi_L\,[E_2]_\alpha)$
- $\Pi_L\,([\lambda\, v_i.\, E]_\alpha) \stackrel{\text{def}}{=} \mathsf{LAM}\, v_i.\, \Pi_{v_i,l}\,([E]_\alpha)$ where we write $\mathsf{LAM}\, v_i.\, \xi$ as an abbreviation for $\mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_i.\, \xi))$.

Π_ϵ is **representationally adequate** in the sense that

Bijective it is bijective (onto its image); and
Commutes with Substitution which means that

$$\Pi_\epsilon\,([E[E'/v_k]]_\alpha) \stackrel{\text{def}}{=} \Pi_\epsilon\,([E]_\alpha[[E']_\alpha/v_k]) = \Pi_\epsilon\,([E']_\alpha)[\Pi_\epsilon\,([E]_\alpha)/\mathsf{VAR}\ k]$$

We give some motivation for the definition of $\mathsf{lbnd}\ n$ in the following remark.

Remark 4.4 We give an informal explanation of the function $\mathsf{lbnd}\ n$. Recall the expression $E_O \stackrel{\text{def}}{=} \lambda\, v_8.\, \lambda\, v_2.\, v_8\ v_3$ from page 3, encoded in Hybrid as $E_H = \mathsf{LAM}\, v_8.\, (\mathsf{LAM}\, v_2.\, (v_8\ \$\$\ \mathsf{VAR}\ 3))$. Now read over the statement of Theorem 4.3 and then compute $\Pi_\epsilon\,[E_O]_\alpha$ which is the formal definition of the Hybrid encoding of E_O. The answer should be the expression E_H (below) where the abbreviations above have been fully expanded. The function Π_ϵ works recursively. λ-binder nodes are replaced by instances of LAM (recall that $\mathsf{LAM}\, v_i.\, \xi$ as an abbreviation for $\mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_i.\, \xi))$). Application nodes are replaced by \$\$. Π_L recursively collects the names of the λ-binders in L, and when it reaches leaf node variables it checks to see if the leaf is in scope of a binder or not. If it is (eg v_8), the leaf remains un-changed; if not, (eg v_3) the leaf v_i

becomes VAR i. Thus E_H is

$$\mathsf{ABS}\ (\mathsf{lbnd}\ 0\ \Lambda\, v_8.\ (\mathsf{ABS}\ (\mathsf{lbnd}\ 0\ \Lambda\, v_2.\ (v_8\ \$\$\ \mathsf{VAR}\ 3))))$$

Now use the definition of lbnd to verify that E_H is equal to the de Bruijn expression ABS (ABS (BND 1 $$ VAR 3)). Informally, each instance of the lbnd function descends recursively through its argument via higher-order matching of the meta-abstractions. At each node, a meta-abstraction is "moved" towards the leaf nodes; and when descending over ABS nodes, a counter is increased. All leaf nodes are left un-changed, unless they are variables. If the name of the meta-abstraction (eg $\Lambda\, v_2.$) which has been "moved" to the leaf (eg v_8) has a different name (eg $\Lambda\, v_2.\, v_8$ from the inner lbnd above), the leaf node remains un-changed (eg v_8). If it has the same name (eg $\Lambda\, v_8.\, v_8$ from the outer lbnd above, once the inner has been computed), the node becomes BND n where n is the counter—thus the original bound name becomes the correct de Bruijn bound index (eg v_8 becomes BND 1). End of remark.

We will prove the theorem later on. For now, we give an outline of the structure of the proof. Consider the diagram

$$\begin{array}{ccc} \mathcal{LE}/{\sim_\alpha} & \underset{\phi_L}{\overset{\theta_L}{\rightleftarrows}} & \mathcal{DB}(|L|) \\ \Big\downarrow{\scriptstyle \Pi_L} & & \Big\downarrow{\scriptstyle \mathsf{inst}\ 0\ L} \\ \Pi_L\,(\mathcal{LE}/{\sim_\alpha}) & & \mathcal{CLF}_{exp}(\Gamma^L_{exp}) \end{array}$$

The proof strategy for Theorem 4.3 for is as follows. The functions θ_L and ϕ_L are defined in Proposition 4.10 but are "essentially" $[\![-]\!]_L$ and $(\!|-|\!)_L$. In Lemma 4.5 we show that there is a function, inst, which maps de Bruijn expressions to canonical forms. In Lemma 4.6 we show the existence of a function from canonical forms to de Bruijn expressions, which is shown to be a left inverse for the inst function in Proposition 4.7. Two further technical lemmas 4.8 and 4.9 yield Proposition 4.10, where we prove that the specified action of Π_L is equal to the action of the composition $(\mathsf{inst}\ 0\ L) \circ \theta_L$ of two well defined functions, showing that Π_L exists as a function. We also prove Proposition 4.13 which shows that inst commutes with substitution. These facts for inst together with Theorem 3.8 allow us to prove Hybrid adequacy. Note that lemmas 4.11 and 4.12 are used to prove Proposition 4.13.

Lemma 4.5 For each $n \geq 0$ and list L, the clauses given below yield a well-defined function with the following source and target

$$\mathsf{inst}\ n\ L\ : \mathcal{DB} \to \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

- $\mathsf{inst}\ n\ L\ \mathsf{con}(\nu) \stackrel{\text{def}}{=} \mathsf{CON}\ \nu$
- $\mathsf{inst}\ n\ L\ \mathsf{var}(i) \stackrel{\text{def}}{=} \mathsf{VAR}\ i$
- $\mathsf{inst}\ n\ L\ \mathsf{bnd}(j) \stackrel{\text{def}}{=} \begin{cases} \mathsf{el}\,(j-n)\,L & \text{if } n \leq j \text{ and } j-n < |L| \\ \mathsf{BND}\ j & \text{otherwise, that is } n > j \text{ or } j-n \geq |L| \end{cases}$
- $\mathsf{inst}\ n\ L\ (D_1\ \$\ D_2) \stackrel{\text{def}}{=} (\mathsf{inst}\ n\ L\ D_1)\ \$\$\ (\mathsf{inst}\ n\ L\ D_2)$
- $\mathsf{inst}\ n\ L\ \mathsf{abs}(D) \stackrel{\text{def}}{=} \mathsf{ABS}\ (\mathsf{inst}\ (n+1)\ L\ D)$

Proof First, note that $\mathsf{el}\,(j-n)\,L$ is defined since $0 \leq j-n < |L|$. Now recall Figure 3. We prove, by induction on D, that for any $n \geq 0$ and any list L,

$$\Gamma^L_{exp} \vdash_{can} \mathsf{inst}\ n\ L\ D :: exp$$

$\boxed{\mathsf{con}(\nu)}$ It is clear that $\Gamma^L_{exp} \vdash_{can} \mathsf{CON}\ \nu :: exp$. The rule CST is used twice.

$\boxed{\mathsf{var}(i)}$ It is clear that $\Gamma^L_{exp} \vdash_{can} \mathsf{VAR}\ i :: exp$. The rule CST is used twice.

$\boxed{\mathsf{bnd}(j)}$ Either it is clear from VAR that $\Gamma^L_{exp} \vdash_{can} \mathsf{el}\,(j-n)\,L :: exp$ or we reason for $\mathsf{BND}\ j$ as in the case of $\mathsf{var}(i)$.

$\boxed{\mathsf{abs}(D)}$ By induction we have $\Gamma^L_{exp} \vdash_{can} \mathsf{inst}\ (n+1)\ L\ D :: exp$. The result is immediate from CST with κ instantiated with ABS.

$\boxed{D_1\ \$\ D_2}$ The easy details are omitted. □

Lemma 4.6 For all $n \geq 0$ and list L, the clauses given below give a well defined function with the following source and target

$$\mathsf{inst}^{-1}\ n\ L : \mathcal{CLF}_{exp}(\Gamma^L_{exp}) \to \mathcal{DB}$$

- $\mathsf{inst}^{-1}\ n\ L\ v_i \stackrel{\text{def}}{=} \mathsf{bnd}((\mathsf{pos}\ v_i\ L) + n)$
- $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{CON}\ \nu) \stackrel{\text{def}}{=} \mathsf{con}(\nu)$
- $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{VAR}\ i) \stackrel{\text{def}}{=} \mathsf{var}(i)$
- $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{BND}\ j) \stackrel{\text{def}}{=} \mathsf{bnd}(j)$
- $\mathsf{inst}^{-1}\ n\ L\ (c_1\ \$\$\ c_2) \stackrel{\text{def}}{=} (\mathsf{inst}^{-1}\ n\ L\ (c_1))\ \$\ (\mathsf{inst}^{-1}\ n\ L\ (c_2))$
- $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{ABS}\ c) \stackrel{\text{def}}{=} \mathsf{abs}(\mathsf{inst}^{-1}\ (n+1)\ L\ c)$

Proof Informally, the existence of the function is obvious, as the clauses specifying the function cover all possible canonical forms. However, the details

do require a little care.

The proof is by induction on derivations. More precisely, we prove

$$(\forall \Gamma \vdash_{can} c :: \sigma)(\forall n \geq 0)(\forall L)(\Gamma = \Gamma^L_{exp} \wedge \sigma = exp \Longrightarrow \mathsf{inst}^{-1}\ n\ L\ c \in \mathcal{DB})$$

VAR Choose any n and L. From the assumptions we have $\Gamma^L_{exp} \vdash_{can} c :: exp$ and so for some list $\vec{c'}$ of canonical forms we must have $c \equiv v_k\, \vec{c'}$. But we must have $\Gamma^L_{exp}(v_k) = exp$, a ground type, and so the list must be empty and $c \equiv v_k$; moreover $v_k \in L$. Hence $\mathsf{inst}^{-1}\ n\ L\ v_k$ is defined and is (trivially) in $\mathcal{DB}$.

CST If $\Gamma^L_{exp} \vdash_{can} c :: exp$ was derived using CST, then for some list $\vec{c'}$ of canonical forms we must have $c \equiv \kappa\, \vec{c'}$. But c has type exp, and so κ can only be CON, VAR, BND, \$\$ or ABS. We examine the details only for VAR. Certainly $c \equiv \mathsf{VAR}\, \vec{c'}$ can only be $\mathsf{VAR}\, c'$. Now, $\Gamma^L_{exp} \vdash_{can} c' :: var$ cannot be deduced from ABS. The other possibilities are conclusions to the rules VAR and CST.

$\Gamma^L_{exp} \vdash_{can} c' :: var$ cannot be deduced from VAR, since if so, for some list $\vec{c''}$ of canonical forms we must have $c' \equiv v_k\, \vec{c''}$. But we must have $\Gamma^L_{exp}(v_k) = exp$, a ground type, and so the list must be empty and further $exp \equiv var$, a contradiction.

$\Gamma^L_{exp} \vdash_{can} c' :: var$ cannot be deduced from CST, since $c' \equiv \kappa\, \vec{c''}$ for a constructor κ. The only possibility is that $\vec{c''}$ is of length 0 and κ is some i, that is $c' = i$. Hence $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{VAR}\ i)$ is defined and is (trivially) in $\mathcal{DB}$.

ABS The easy details are omitted—just note that the assumption $\sigma = exp$ is always false, as σ is forced to be a higher type. □

Proposition 4.7 Given any $D \in \mathcal{DB}$, $n \geq 0$ and ordered list L, we have

$$\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ D) = D$$

In particular, each function

$$\mathsf{inst}\ n\ L\ : \mathcal{DB} \to \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

is injective, with left sided inverse $\mathsf{inst}^{-1}\ n\ L$.

Proof The existence of the functions follows from Lemmas 4.5 and 4.6. The equalities are proved by induction over de Bruijn expressions.

$\boxed{\mathsf{con}(\nu)}$ $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ \mathsf{con}(\nu)) = \mathsf{inst}^{-1}\ n\ L\ (\mathsf{CON}\ \nu) = \mathsf{con}(\nu)$.

$\boxed{\mathsf{var}(i)}$ $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ \mathsf{var}(i)) = \mathsf{inst}^{-1}\ n\ L\ (\mathsf{VAR}\ i) = \mathsf{var}(i)$.

$\boxed{\mathsf{bnd}(j)}$ The first possibility is that

$$\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ j) = \mathsf{inst}^{-1}\ n\ L\ (\mathsf{el}\ (j-n)\ L) = \mathsf{bnd}((\mathsf{pos}\ (\mathsf{el}\ (j-n)\ L)\ L) + n) = \mathsf{bnd}(j)$$

Note that this depends crucially on the fact that L is ordered.

For the second, we have $\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ j) = \mathsf{inst}^{-1}\ n\ L\ (\mathsf{BND}\ j) = \mathsf{bnd}(j)$.

$\boxed{\mathsf{abs}(D)}$

$$\begin{aligned}\mathsf{inst}^{-1}\ n\ L\ (\mathsf{inst}\ n\ L\ \mathsf{abs}(D)) &= \mathsf{inst}^{-1}\ n\ L\ (\mathsf{ABS}\ (\mathsf{inst}\ (n+1)\ L\ D)) \\ &= \mathsf{abs}(\mathsf{inst}^{-1}\ (n+1)\ L\ (\mathsf{inst}\ (n+1)\ L\ D)) \\ &= \mathsf{abs}(D)\end{aligned}$$

where the final equality follows by induction. The details for applications are straightforward and omitted. □

Lemma 4.8 Given any $D \in \mathcal{DB}$, list L, $n \geq 0$, and metavariable v_k fresh for L, we have

$$\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ D) = \mathsf{inst}\ (n+1)\ L\ D$$

Proof We prove this by induction on D. The argument for constants $\mathsf{con}(\nu)$ is similar to that for the inductive case of free indices:

$\boxed{\mathsf{var}(i)}$

$$\begin{aligned}&\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ \mathsf{var}(i)) \\ &\qquad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{VAR}\ i) \\ &\qquad = \mathsf{VAR}\ i \\ &\qquad = \mathsf{inst}\ (n+1)\ L\ \mathsf{var}(i)\end{aligned}$$

The equations follow from routine calculations.

$\boxed{\mathsf{bnd}(j)}$ Suppose that $n \leq j$ and $j - n < |v_k, L|$. We consider separately the cases of $j - n = 0$ and $j - n \geq 1$. If $j - n = 0$ then

$$\begin{aligned}
&\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(j)) \\
&\qquad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{el}\ (j - n)\ (v_k, L)) \\
&\qquad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, v_k) \\
&\qquad = \mathsf{BND}\ n = \mathsf{BND}\ j \\
&\qquad = \mathsf{inst}\ (n + 1)\ L\ \mathsf{bnd}(j)
\end{aligned}$$

where the final equality follows since $n + 1 > n = j$. If $j - n \geq 1$ then

$$\begin{aligned}
&\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(j)) \\
&\quad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, (\mathsf{el}\ (j - n)\ (v_k, L))) \\
&\quad = \mathsf{lbnd}\ n\ \Lambda\ (v_k.\, v_{k'}) \qquad\qquad \textit{where } k \neq k' \textit{ by freshness} \\
&\quad = v_{k'} \\
&\quad = \mathsf{el}\ (j - (n + 1))\ L \\
&\quad = \mathsf{inst}\ (n + 1)\ L\ \mathsf{bnd}(j)
\end{aligned}$$

where the final equality follows since $n + 1 \leq j$ and $j - (n + 1) < |L|$.

Now suppose that either $n > j$ or $j - n \geq |v_k, L|$.

$$\begin{aligned}
&\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(j)) \\
&\qquad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{BND}\ j) \\
&\qquad = \mathsf{BND}\ j \\
&\qquad = \mathsf{inst}\ (n + 1)\ L\ \mathsf{bnd}(j)
\end{aligned}$$

where the final equality follows since either $n + 1 > j$ or $j - (n + 1) \geq |L|$.

$\boxed{\mathsf{abs}(D)}$

$$\begin{aligned}
&\mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{inst}\ n\ (v_k, L)\ \mathsf{abs}(D)) \\
&\qquad = \mathsf{lbnd}\ n\ (\Lambda\, v_k.\, \mathsf{ABS}\ (\mathsf{inst}\ (n+1)\ (v_k, L)\ D)) \\
&\qquad = \mathsf{ABS}\ (\mathsf{lbnd}\ (n+1)\ (\Lambda\, v_k.\, \mathsf{inst}\ (n+1)\ (v_k, L)\ D)) \\
&\qquad = \mathsf{ABS}\ (\mathsf{inst}\ (n+2)\ L\ D) \\
&\qquad = \mathsf{inst}\ (n+1)\ L\ \mathsf{abs}(D)
\end{aligned}$$

The first, second and fourth equalities are true by definition. The third is by induction.

$\boxed{D_1\ \$\ D_2}$ The details are omitted, and are similar to the previous cases.

□

Lemma 4.9 For all $E \in \mathcal{LE}$, and all $n \geq 0$, lists L, L' such that $n = |L'|$, and variables v_i, v_k we have

$$(\mathsf{inst}\ n\ (v_i, L)\ [\![E]\!]_{L', v_i, L})[v_k/v_i] = \mathsf{inst}\ n\ (v_k, L)\ [\![E]\!]_{L', v_i, L}$$

Proof The proof is by induction over $E \in \mathcal{LE}$. The details for constants and applications are easy and omitted.

$\boxed{v_\alpha}$

(Case $v_\alpha \notin L', v_i, L$): $[\![v_\alpha]\!]_{L', v_i, L} = \mathsf{var}(\alpha)$ and so

$$\begin{aligned}
(\mathsf{inst}\ n\ (v_i, L)\ [\![v_\alpha]\!]_{L', v_i, L})[v_k/v_i] &= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{var}(\alpha))[v_k/v_i] \\
&= (\mathsf{VAR}\ \alpha)[v_k/v_i] \\
&= (\mathsf{VAR}\ \alpha) \\
&= \mathsf{inst}\ n\ (v_k, L)\ \mathsf{var}(\alpha)
\end{aligned}$$

(Case $v_\alpha \in L'$): $[\![v_\alpha]\!]_{L', v_i, L} = \mathsf{bnd}(j)$ where $j = \mathsf{pos}\, v_\alpha\, (L', v_i, L) < |L'| = n$

$$\begin{aligned}
(\mathsf{inst}\ n\ (v_i, L)\ [\![v_\alpha]\!]_{L', v_i, L})[v_k/v_i] &= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{bnd}(j))[v_k/v_i] \\
&= \mathsf{BND}\ j \\
&= \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(j)
\end{aligned}$$

(Case $v_\alpha \notin L'$ and $\alpha = i$): $[\![v_\alpha]\!]_{L',v_i,L} = \mathsf{bnd}(\mathsf{pos}\, v_\alpha\, (L', v_i, L)) = \mathsf{bnd}(n)$

$$\begin{aligned}(\mathsf{inst}\ n\ (v_i, L)\ [\![v_\alpha]\!]_{L',v_i,L})[v_k/v_i] &= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{bnd}(n))[v_k/v_i] \\ &= v_i[v_k/v_i] \\ &= v_k \\ &= \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(n)\end{aligned}$$

(Case $v_\alpha \notin L', v_i$ and $v_\alpha \in L$): $[\![v_\alpha]\!]_{L',v_i,L} = \mathsf{bnd}(j)$ where

$$j = \mathsf{pos}\, v_\alpha\, (L', v_i, L) > n$$

and so $0 < j - n = \mathsf{pos}\, v_\alpha\, (v_i, L) < |v_i, L|$

$$\begin{aligned}(\mathsf{inst}\ n\ (v_i, L)\ [\![v_\alpha]\!]_{L',v_i,L})[v_k/v_i] &= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{bnd}(j))[v_k/v_i] \\ &= (\mathsf{el}\ (j-n)\ (v_i, L))[v_k/v_i] \\ &= v_\alpha[v_k/v_i] \\ &= v_\alpha \\ &= \mathsf{el}\ (j-n)\ (v_k, L) \\ &= \mathsf{inst}\ n\ (v_k, L)\ \mathsf{bnd}(j)\end{aligned}$$

$\boxed{\lambda\, v_\alpha.\, E}$ We have

$$\begin{aligned}&(\mathsf{inst}\ n\ (v_i, L)\ [\![\lambda\, v_\alpha.\, E]\!]_{L',v_i,L})[v_k/v_i] \\ &\quad= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{abs}([\![E]\!]_{v_\alpha,L',v_i,L}))[v_k/v_i] \\ &\quad= \mathsf{ABS}\ (\mathsf{inst}\ (n+1)\ (v_i, L)\ [\![E]\!]_{v_\alpha,L',v_i,L}) \\ &\quad= \mathsf{ABS}\ ((\mathsf{inst}\ (n+1)\ (v_k, L)\ [\![E]\!]_{v_\alpha,L',v_i,L})[v_k/v_i]) \\ &\quad= (\mathsf{inst}\ n\ (v_i, L)\ \mathsf{abs}([\![E]\!]_{v_\alpha,L',v_i,L}))[v_k/v_i] \\ &\quad= \mathsf{inst}\ n\ (v_k, L)\ [\![\lambda\, v_\alpha.\, E]\!]_{L',v_i,L}\end{aligned}$$

with the third equality following by induction. □

Proposition 4.10 For any $[E]_\alpha \in \mathcal{LE}/\!\sim_\alpha$ and list L we have

$$\Pi_L\, [E]_\alpha = \mathsf{inst}\ 0\ L\ (\theta_L([E]_\alpha))$$

where the action of Π_L is specified in the statement of Theorem 4.3, $\phi_L \stackrel{\text{def}}{=} q \circ (\!|-|\!)_L$ where q is the α-equivalence quotient function, and for any E we set $\theta_L([E]_\alpha) \stackrel{\text{def}}{=} [\![E]\!]_L$ well defined by Proposition A.13.

Hence the action specifies a well defined function Π_L as it is the composition action of two other well defined functions; the source and target of Π_L are given below.

Further, there is a function Σ_L whose action is equal to that of $\mathsf{inst}\ 0\ L$ and such that it furnishes a factorisation through the inclusion given below

$$\begin{array}{ccccc}
\mathcal{LE}/\!\sim_\alpha & \underset{\phi_L}{\overset{\theta_L}{\rightleftarrows}} & \mathcal{DB}(|L|) & = & \mathcal{DB}(|L|) \\
\Big\downarrow \Pi_L & & \Big\downarrow \Sigma_L & & \Big\downarrow \mathsf{inst}\ 0\ L \\
\Pi_L\,(\mathcal{LE}/\!\sim_\alpha) & = & \Pi_L\,(\mathcal{LE}/\!\sim_\alpha) & \hookrightarrow & \mathcal{CLF}_{exp}(\Gamma^L_{exp})
\end{array}$$

and moreover if L is ordered then Σ_L is injective.

Proof Note that the function $\mathsf{inst}\ 0\ L$ is defined on $\mathcal{DB}$ and hence on any subset!

We prove by induction on $E \in \mathcal{LE}$,

$$(\forall E)(\forall L)(\mathsf{inst}\ 0\ L\ (\theta_L([E]_\alpha)) = \Pi_L\,([E]_\alpha))$$

$\boxed{\nu}$ $\mathsf{inst}\ 0\ L\ (\theta_L([\nu]_\alpha)) \stackrel{\text{def}}{=} \mathsf{CON}\ \nu \stackrel{\text{def}}{=} \Pi_L\,([\nu]_\alpha)$.

$\boxed{v_i}$ When $v_i \in L$ we have

$$\mathsf{inst}\ 0\ L\ (\theta_L([v_i]_\alpha)) = \mathsf{inst}\ 0\ L\ \mathsf{bnd}(\mathsf{pos}\,v_i\,L) = \mathsf{el}\,(\mathsf{pos}\,v_i\,L)\,L = v_i \stackrel{\text{def}}{=} \Pi_L\,([v_i]_\alpha)$$

where the second equality follows since $0 \leq \mathsf{pos}\,v_i\,L < |L|$ and the other simple case is left to the reader.

$\boxed{E_1\ E_2}$ A simple exercise.

$\boxed{\lambda\, v_i.\, E}$ We choose v_k to be fresh for L, E and v_i. Then

$$
\begin{aligned}
&\mathsf{inst}\ 0\ L\ (\theta_L([\lambda\, v_i.\, E]_\alpha)) \\
(\text{Def } \theta_L \text{ and A.13}) \quad &= \mathsf{inst}\ 0\ L\ (\theta_L([\lambda\, v_k.\, E[v_k/v_i]]_\alpha)) \\
&\stackrel{\text{def}}{=} \mathsf{inst}\ 0\ L\ (\mathsf{abs}([\![E[v_k/v_i]]\!]_{v_k,L})) \\
&\stackrel{\text{def}}{=} \mathsf{ABS}\ (\mathsf{inst}\ 1\ L\ [\![E[v_k/v_i]]\!]_{v_k,L}) \\
(4.8 \text{ where } v_k \notin L) \quad &= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_k.\, \mathsf{inst}\ 0\ (v_k, L)\ [\![E[v_k/v_i]]\!]_{v_k,L})) \\
(\text{A.12 where } v_k \notin fv(E)) \quad &= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_k.\, \mathsf{inst}\ 0\ (v_k, L)\ [\![E]\!]_{v_i,L})) \\
(4.9) \quad &= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_k.\, (\mathsf{inst}\ 0\ (v_i, L)\ [\![E]\!]_{v_i,L})[v_k/v_i])) \\
(4.2) \quad &= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_i.\, \mathsf{inst}\ 0\ (v_i, L)\ [\![E]\!]_{v_i,L})) \\
&= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_i.\, \mathsf{inst}\ 0\ (v_i, L)\ \theta_{v_i,L}([E]_\alpha))) \\
&= \mathsf{ABS}\ (\mathsf{lbnd}\ 0\ (\Lambda\, v_i.\, \Pi_{v_i,L}\ [E]_\alpha)) \\
&\stackrel{\text{def}}{=} \Pi_L\ ([\lambda\, v_i.\, E]_\alpha)
\end{aligned}
$$

In appealing to Lemma 4.2 we must ensure that $v_k \notin \mathsf{inst}\ 0\ (v_i, L)\ [\![E]\!]_{v_i,L}$. In fact it is easy to see that for any D, $\hat{L}$ and m, the variables occuring in $\mathsf{inst}\ m\ \hat{L}\ D$ must come from $\hat{L}$. Thus the condition holds as v_k is fresh for v_i, L. The penultimate equality is by induction.

It follows that we can define Σ_L by taking its action to be that of $\mathsf{inst}\ 0\ L$. Given $D \in \mathcal{DB}(|L|)$ we have $D = \theta_L(\phi_L(D))$ by Proposition A.1 and so

$$\Sigma_L\ (D) \stackrel{\text{def}}{=} \mathsf{inst}\ 0\ L\ D = \mathsf{inst}\ 0\ L\ (\theta_L(\phi_L(D))) = \Pi_L\ (\phi_L(D)) \in \Pi_L\ (\mathcal{LE}/{\sim_\alpha})$$

and hence by Proposition 4.7 we deduce Σ_L is injective in the case that L is ordered.

□

Before stating the adequacy theorem, we must define a substitution function for Hybrid.

Lemma 4.11 There is a function

$$\mathcal{CLF}_{exp}(\Gamma^L_{exp}) \times \mathcal{CLF}_{exp}(\Gamma^L_{exp}) \times \mathbb{N} \to \mathcal{CLF}_{exp}(\Gamma^L_{exp})$$

denoted by $(c, c', k) \mapsto c[c'/\mathsf{VAR}\ k]$, which, informally, maps (c, c', k) to the expression c in which all occurrences of $\mathsf{VAR}\ k$ are replaced by c'.

Proof The definition of the function, and the proof, are omitted. Note that the set $\mathcal{CLF}_{exp}(\Gamma^L_{exp})$ does not involve expressions which bind variables, and so the definition of the function is entirely straightforward. □

Lemma 4.12 For all $n \geq 0$, ordered L, and $D \in \mathcal{DB}(|L|)$, if $n \geq m$ where m is the minimum level of D, then

$$\mathsf{inst}\ (n+1)\ L\ D = \mathsf{inst}\ n\ L\ D$$

Proof This is a routine induction over D. In the case that D is $\mathsf{bnd}(j)$, note that inst returns $\mathsf{BND}\ j$ as the minimum level m of $\mathsf{bnd}(j)$ is $j+1$, and thus both n and $n+1$ are strictly greater than j. □

Proposition 4.13 For any $D, D' \in \mathcal{DB}$, $n, k \geq 0$, and ordered L, if $n \geq m$ where m is the minimum level of D', then

$$\mathsf{inst}\ n\ L\ (D[D'/\mathsf{var}(k)]) = (\mathsf{inst}\ n\ L\ D)[\mathsf{inst}\ n\ L\ D'/\mathsf{VAR}\ k]$$

Proof The proof is by induction on D. All of the cases are easy, except for abstractions $\mathsf{abs}(D)$ which require Lemma 4.12. We have

$$\begin{aligned}
&\mathsf{inst}\ n\ L\ (\mathsf{abs}(D)[D'/\mathsf{var}(k)]) \\
&\qquad = \mathsf{ABS}\ (\mathsf{inst}\ (n+1)\ L\ (D[D'/\mathsf{var}(k)])) \\
&\qquad = \mathsf{ABS}\ ((\mathsf{inst}\ (n+1)\ L\ D)[\mathsf{inst}\ (n+1)\ L\ D'/\mathsf{VAR}\ k]) \\
&\qquad = (\mathsf{inst}\ n\ L\ \mathsf{abs}(D))[\mathsf{inst}\ (n+1)\ L\ D'/\mathsf{VAR}\ k]
\end{aligned}$$

The second equality is by induction, as $n+1 \geq n \geq m$ where m is the minimum level of D'. The third follows from simple applications of the definitions, and we are done by appeal to Lemma 4.12 applied to D'. □

We can now prove Theorem 4.3.

Proof

Bijective We only need to show that Π_ϵ is injective. From Proposition 4.10 we know that $\Pi_\epsilon = \Sigma_\epsilon \circ \theta_\epsilon$. But $\theta = \theta_\epsilon$ is injective by Theorem 3.8 and Σ_ϵ is injective by Proposition 4.10 (noting that ϵ is ordered by definition).

Commutes with Substitution We calculate

$$\begin{aligned}
\Pi_\epsilon\ ([E]_\alpha[[E']_\alpha/v_k]) &= \mathsf{inst}\ 0\ \epsilon\ [\![E[E'/v_k]]\!]_\epsilon && (11)\\
&= \mathsf{inst}\ 0\ \epsilon\ ([\![E]\!]_\epsilon[[\![E']\!]_\epsilon/\mathsf{var}(k)]) && (12)\\
&= (\mathsf{inst}\ 0\ \epsilon\ [\![E]\!]_\epsilon)[\mathsf{inst}\ 0\ \epsilon\ [\![E']\!]_\epsilon/\mathsf{VAR}\ k] && (13)\\
&= (\Pi_\epsilon\ [E]_\alpha)[\Pi_\epsilon\ [E']_\alpha/\mathsf{VAR}\ k] && (14)
\end{aligned}$$

Equation 11 follows from Proposition 4.10. Equation 12 follows from Theorem 3.8. Equation 13 follows from Proposition 4.13—note that the value of n in the lemma is 0 and so $n \geq m$ where $m = 0$ is the minimum level of $[\![E']\!]_\epsilon \in \mathcal{DB}(0)$. Equation 14 follows from Proposition 4.10. □

5 Conclusions, Further and Related Work

We have sometimes found that it can take time to explain how Hybrid works, and moreover how it is applied in practice, when dealing only with the actual Isabelle HOL implementation. One reason for this is the not inconsiderable number of forms of abstraction and quantification that one has to deal with. Thus we hope that apart from presenting an interesting key result and proof in its own right, this paper is seen to present a clean description of Hybrid which will be useful to potential users. When dealing with the implementation, it is not so easy to see transparently how the pure λ-calculus resides in the system—here, it is encoded very directly!

A topic for future work concerns adequacy results for object level logics [2]. For this to be of use, the encoding must of course be adequate. We believe that we can regard the results of this paper as a form of *generic adequacy* for object logics. Roughly speaking, in order to prove that an object level encoding into Hybrid is adequate, we can factor the encoding through $\mathcal{LE}/\!\sim_\alpha$. The encoding into $\mathcal{LE}/\!\sim_\alpha$ is proved adequate in the usual way [21]; and the Hybrid encoding must then be adequate by post-composition with the adequacy function presented in this paper.

We also want to look in greater detail at how Hybrid represents certain object level expressions. For example, in Hybrid there is a predicate abstr which detects if an expression is equal up to η-equality to an expression $\mathsf{LAM}\, v_i.\, e\, v_i$. Thus, from the perspective of the current paper, we will look at representation results such as

Theorem 5.1 Suppose that $\mathsf{abstr}\, e$ holds. Then there exists $[\lambda\, v_i.\, E]_\alpha \in \mathcal{LE}/\!\sim_\alpha$ such that $\Pi_\epsilon\, ([\lambda\, v_i.\, E]_\alpha) = e$.

Our work could not have come about without the contributions of others. We briefly review other ways of dealing with variable binding.

- For origins of de Bruijn syntax see [5].
- Some authors have presented concrete name-carrying syntax: abstractions are pairs "(name, expression)" and the mechanization works directly on first-order parse trees, which are quotiented by α-conversion [18,26]. While recursion/induction is well-supported, the amount of detail that needs to

be taken care of on a case-by-case basis tends to be overwhelming.
- In [9,23] Gabbay and Pitts have introduced a novel approach, based on the remarkable observation that a first-order theory of α-conversion and binding is better founded on the notion of name *swapping* rather than renaming. For recent related work, see [24,25].
- Abstractions as functions from *names* to expressions: mentioned in [4] (and developed in [10]) it was first proposed in [6], as a way to have binders as functions on inductive data-types, while coping with the issue of *exotic* expressions stemming from an inductive characterization of the set of names. See also Honsell et al.'s framework [13], which explicitly embraces an *axiomatic* approach to meta-reasoning with HOAS.
- Abstractions as functions from *expressions* to expressions are presented in [22,11]. Related work, where the emphasis is on trying to allow (primitive) recursive definitions on functions of higher type while preserving adequacy of representations, has been realized for the simply-typed case in [8] and more recently for the dependently-typed case in [7].

A A Bijection Between λ-calculus and Proper de Bruijn

We give a (pencil and paper) sketch proof of the often quoted but seldom proved theorem given below. There are a number of reasons why we have chosen to give a very careful proof in this paper:

- Readers can become familiar with notation which is employed in the proof of adequacy for Hybrid.
- The proofs are delicate, requiring care to ensure their correctness. This section provides a full and sound basis for our proof of adequacy.
- Our proof of the bijection takes *great care* over the notion of α-equivalence; in particular, we construct the bijection by beginning with functions defined on genuine λ-calculus expressions, with no assumptions at all about variable conventions concerning binding and bound variables.

In order to prove Theorem 3.1, we establish the existence of the family of pairs of functions

$$[\![-]\!]_L : \mathcal{LE} \rightleftarrows \mathcal{DB}(|L|) : (\!|-|\!)_L$$

whose definitions were given in Proposition 3.2 and Proposition 3.3 and whose proofs follow below. In Propositions A.1 and A.10 we prove that each pair of maps "almost" gives rise to an isomorphism (the intermediate lemmas are used to prove A.10) . Making use of Lemma A.11 and Lemma A.12, we show in Proposition A.13 that the function $[\![-]\!]_\epsilon$ is well-defined on $\mathcal{LE}/\!\sim_\alpha$. We then prove Theorem 3.1.

Proof of Proposition 3.2.

Proof One first proves by induction on E,

$$(\forall E \in \mathcal{LE})(\forall L)([\![E]\!]_L \in \mathcal{DB})$$

(which is virtually immediate) so that uses of level type check, and then one can prove

$$(\forall E \in \mathcal{LE})(\forall L)(\mathsf{level}\,|L|\,[\![E]\!]_L).$$

We give details of the proof. Each inductive case is indicated using a $\boxed{}$, and within each case L is an arbitrary list.

$\boxed{\nu}$ $\mathsf{level}\,|L|\,[\![\nu]\!]_L = \mathsf{level}\,|L|\,\nu = T.$

$\boxed{v_i}$ If $v_i \notin L$, which includes the case when L is empty,

$$\mathsf{level}\,|L|\,[\![v_i]\!]_L = \mathsf{level}\,|L|\,\mathsf{var}(i) = T$$

If $v_i \in L$,

$$\mathsf{level}\,|L|\,[\![v_i]\!]_L = \mathsf{level}\,|L|\,(\mathsf{pos}\,v_i\,L) = (\mathsf{pos}\,v_i\,L) < |L| = T$$

$\boxed{\lambda\, v_i.\, E}$

$$\mathsf{level}\,|L|\,[\![\lambda\, v_i.\, E]\!]_L = \mathsf{level}\,|L|\,\mathsf{abs}([\![E]\!]_{v_i,L}) = \mathsf{level}\,(|L|+1)\,[\![E]\!]_{v_i,L} = T$$

with the final equality holding by induction.

$\boxed{E_1\ E_2}$ The easy details are omitted.

□

Proof of Proposition 3.3.

Proof

One proves by induction on D

$$(\forall D \in \mathcal{DB})(\forall \text{ ordered } L)(\mathsf{level}\,|L|\,D \Longrightarrow (\!|D|\!)_L \in \mathcal{LE})$$

$\boxed{\mathsf{con}(\nu)}$ Note that $(\!|\mathsf{con}(\nu)|\!)_L = \nu$ which is always in $\mathcal{LE}$.

$\boxed{\mathsf{var}(i)}$ Note that $(\!|\mathsf{var}(i)|\!)_L = v_i$ which is always in $\mathcal{LE}$.

$\boxed{\mathsf{bnd}(j)}$ If $\mathsf{level}\,|L|\,\mathsf{bnd}(j)$ then $0 \leq j < |L|$ so that $L \neq \epsilon$. Hence $(\!|\mathsf{bnd}(j)|\!)_L = \mathsf{el}\,j\,L$ is *defined* and hence exists in $\mathcal{LE}$.

$\boxed{\mathsf{abs}(D)}$ Note that $\mathsf{level}\,|L|\,\mathsf{abs}(D) = \mathsf{level}\,(|L|+1)\,D$. Hence by induction, for any ordered list L', $(\!|D|\!)_{L'} \in \mathcal{LE}$ if $|L'| = |L|+1$. If $M = \mathsf{Max}(D;L)$, then v_{M+1}, L is ordered. Hence $(\!|D|\!)_{v_{M+1},L}$ is in $\mathcal{LE}$, and thus so is $\lambda\, v_{M+1}.\,(\!|D|\!)_{v_{M+1},L}$.

$\boxed{D_1\ \$\ D_2}$ The easy details are omitted.

□

Note that the choice of M in $\lambda\, v_{M+1}.\,(\!|D|\!)_{v_{M+1},L}$ ensures that v_{M+1}, L is ordered, so the recursive definition makes sense, and moreover the binding variable is chosen so that when free indices $\mathsf{var}(i)$ in D are mapped recursively to λ-calculus variables v_i which are in the scope of the binding variable v_{M+1}, they will not be (accidently) captured, as $M+1 > i$. Here is an example which should be checked as an exercise.

$$(\!|\mathsf{abs}(\mathsf{abs}(\mathsf{bnd}(0))\ \$\ \mathsf{abs}(\mathsf{bnd}(3))\ \$\ \mathsf{var}(8))|\!)_{v_7,v_6} = \lambda\, v_9.\,(\lambda\, v_{10}.\, v_{10})\ (\lambda\, v_{10}.\, v_6)\ v_8$$

Proposition A.1 Let $D \in \mathcal{DB}$, and L be any ordered list such that for all $v_k \in L$ if any, $k \geq \mathsf{Max}(D;\epsilon)+1$. Then

$$\mathsf{level}\,|L|\,D \Longrightarrow [\![(\!|D|\!)_L]\!]_L = D$$

Proof A straightforward induction over $D \in \mathcal{DB}$. Constants are trivial. Note that the other two base cases make crucial use of the assumptions in the proposition.

$\boxed{\mathsf{var}(i)}$ We have $[\![(\!|\mathsf{var}(i)|\!)_L]\!]_L = [\![v_i]\!]_L = \mathsf{var}(i)$ for all L, for if $v_i \in L$ then $i \geq \mathsf{Max}(\mathsf{var}(i);\epsilon)+1 = i+1$, a contradiction.

$\boxed{\mathsf{bnd}(j)}$ We have

$$[\![(\!|\mathsf{bnd}(j)|\!)_L]\!]_L = \mathsf{bnd}(\mathsf{pos}\,(\mathsf{el}\,j\,L)\,L) = \mathsf{bnd}(j)$$

because (crucially) L is ordered. The details for the two inductive cases, abstraction and application, are easy and omitted. □

Recall that our next step is to prove Proposition A.10. The proof is not difficult, but it is very delicate, and we require some machinery to deal with substitutions and the re-naming of variables to avoid capture. In particular, in order to carry out many proofs, we need a definition of *simultaneous* substitution. Further, because the function $[\![-]\!]_L$ involves *arbitrary* lists L, we require a definition which mirrors this.

Let L and $L^{\mathcal{LE}}$ be lists of equal length, where L is a list of variables v_k as usual, and $L^{\mathcal{LE}}$ is a list of $\mathcal{LE}$ expressions. Suppose that $E' \in L^{\mathcal{LE}}$ and $v_k \in L$ both occur at some position p. Then we shall call the expression and variable **mates**, and say that one is the **mate** of the other. If $v_k \in L$, then we refer to the first occurrence as **active**, written $v_k{\uparrow}$. Any other occurrences are referred to as **inactive**, written $v_k{\downarrow}$. We write $E[L^{\mathcal{LE}}/L]$ for, informally, the simultaneous capture avoiding substitution of each expression $E' \in L^{\mathcal{LE}}$ for free occurrences in E of its mate in L. The definition of the simpler $E[E'/v_k]$ is then immediate. Note that some free variables in E may have multiple occurrences in L; if so, the expression in $L^{\mathcal{LE}}$ which is the mate of the active occurrence is the one that is substituted—see the formal definition below. For example $(v_1\ v_2)[E_6, E_5, E_8/v_1, v_2, v_1] = E_6\ E_5$.

We want to define such substitutions to be functions on syntax. This will give us a clean and direct definition of α-equivalence. We have to take great care with the definition of capture avoiding substitution on abstractions where a re-naming takes place, in particular with the choice of the re-naming variable. The definition is in Figure A.1.

Proposition A.2 Suppose that $E \sim_\alpha E'$ for any expressions E and E'. Then for any lists L and $L^{\mathcal{LE}}$,

$$E[L^{\mathcal{LE}}/L] \sim_\alpha E'[L^{\mathcal{LE}}/L]$$

Proof The proof is by induction over $\sim_\alpha$. A full proof is, however, surprisingly tricky. In fact many authors gloss over the details, and even suggest that the induction is routine. The proof can only be regarded as routine once a number of other small results have been proven, each one formalizing a fact about properties of simultaneous substitution. Moreover, the "proofs" of each of the results referred to requires the "other" results in its own proof. The upshot is that they must all be proven by induction at the same time, with the proof of each result calling the inductive hypotheses of the others. We collect together these required results in Lemma A.3 and Lemma A.4. □

Lemma A.3 In this lemma, we will regard lists of variable L as lists of

$$\nu[L^{\mathcal{LE}}/L] \stackrel{\text{def}}{=} \nu$$

$$v_k[L^{\mathcal{LE}}/L] \stackrel{\text{def}}{=} \begin{cases} \mathsf{el}\,(\mathsf{pos}\,v_k\,L)\,L^{\mathcal{LE}} \text{ if } v_k \in L \\ v_k \text{ if } v_k \notin L \end{cases}$$

$$(E_1\ E_2)[L^{\mathcal{LE}}/L] \stackrel{\text{def}}{=} (E_1[L^{\mathcal{LE}}/L])\ (E_2[L^{\mathcal{LE}}/L])$$

$$(\lambda\, v_k.\, E)[L^{\mathcal{LE}}/L] \stackrel{\text{def}}{=} \begin{cases} \lambda\, v_k.\, E[\overline{L^{\mathcal{LE}}}/\overline{L}] \\ \qquad \text{if } (\forall X \in \overline{L}) \begin{pmatrix} X{\downarrow} \vee X \notin fv(E) \\ \vee \\ (v_k \notin fv(\mathsf{el}\,(\mathsf{pos}\,X\,\overline{L})\,\overline{L^{\mathcal{LE}}})) \end{pmatrix} \\ \lambda\, v_w.\, E[v_w/v_k][\overline{L^{\mathcal{LE}}}/\overline{L}] \\ \qquad \text{if } (\exists X \in \overline{L}) \begin{pmatrix} X{\uparrow} \wedge X \in fv(E) \\ \wedge \\ v_k \in fv(\mathsf{el}\,(\mathsf{pos}\,X\,\overline{L})\,\overline{L^{\mathcal{LE}}}) \end{pmatrix} \end{cases}$$

where

- w is chosen to be the maximum of the indices occurring E, $L^{\mathcal{LE}}$ and L, plus 1; note that as $v_k \in fv(\mathsf{el}\,(\mathsf{pos}\,X\,\overline{L})\,\overline{L^{\mathcal{LE}}})$ holds in the clause involving w, then $w > k$; and
- given lists $L^{\mathcal{LE}}$ and L of equal length, $\overline{L^{\mathcal{LE}}}$ and $\overline{L}$ are the same lists in which any occurrences of v_k in L together with their mates in $L^{\mathcal{LE}}$ are removed.

Fig. A.1. Simultaneous Substitution

(simple) expressions $L^{\mathcal{LE}}$. Suppose that

$$\begin{aligned} \Phi(E) \stackrel{\text{def}}{=}\ & (\forall L_1, L_1', L_2, L_2') \\ & (L_1 \cap L_2 = \varnothing \wedge L_1 \cap L_2' = \varnothing \Longrightarrow \\ & \qquad E[L_1'/L_1][L_2'/L_2] \sim_\alpha E[L_2'/L_2][L_1'[L_2'/L_2]/L_1]) \end{aligned}$$

and

$$\Phi(E) \stackrel{\text{def}}{=} (\forall L, L', M, M')(\forall v_k) \\ ((\forall X \in L(X{\downarrow} \vee X \notin fvE \vee \mathsf{mate}(X) \neq v_k)) \Longrightarrow \\ E[L'[M'/M]/L] \sim_\alpha E[L'[\overline{M'}/\overline{M}]/L])$$

and

$$\Theta(E) \stackrel{\text{def}}{=} (\forall L, L')(\forall v_k)(v_k \notin fvE \Longrightarrow E[L'/L] \sim_\alpha E[\overline{L'}/\overline{L}])$$

Then for all $E \in \mathcal{LE}$ we have $\Phi(E) \wedge \Psi(E) \wedge \Theta(E)$

Proof The proof is a very tedious strong induction over the size of expressions and is omitted. In verifying, for example, an inductive step for $\Phi(E)$, one typically not only requires inductive hypotheses $\Phi(E')$ but also $\Psi(E')$ and $\Theta(E')$. The three conjuncts cannot be proven independently. □

Lemma A.4 For any $E \in \mathcal{LE}$, any L_1 and $L_1^{\mathcal{LE}}$ of equal length, and L_2 and $L_2^{\mathcal{LE}}$ of equal length, such that no free variable in $L_1^{\mathcal{LE}}$ occurs in L_2, then

$$E[L_1^{\mathcal{LE}}/L_1][L_2^{\mathcal{LE}}/L_2] \sim_\alpha E[L_1^{\mathcal{LE}}, L_2^{\mathcal{LE}}/L_1, L_2]$$

Proof The proof is by induction over the size of E. We omit the proof, but remark that Lemma A.3 is crucial. □

Corollary A.5 The simultaneous substitution function for $\mathcal{LE}$ can be extended to $\mathcal{LE}/\sim_\alpha$. More precisely there is a well defined function specified by $([E]_\alpha, [L^{\mathcal{LE}}]_\alpha, L) \mapsto [E[L^{\mathcal{LE}}/L]]_\alpha$ where $[L^{\mathcal{LE}}]_\alpha$ means a list of α-equivalence classes of expressions.

Proof The idea is to combine Proposition A.2 with the fact (provable by induction) that if $L_1^{\mathcal{LE}}$ and $L_2^{\mathcal{LE}}$ are two lists of equal length and consisting of pairwize α-equivalent expressions, then the function $-[L_1^{\mathcal{LE}}/+]$ equals $-[L_2^{\mathcal{LE}}/+]$. □

Remark A.6 In the remainder of this section, in any $E[L'/L]$ the list L' will in fact be a list of variables.

Lemma A.7 Let $D \in \mathcal{DB}(|L|)$ be any expression, with L any ordered list. Suppose also that $k \geq \mathsf{Max}(D; L) + 1$. Then v_k is not free in $(\!|D|\!)_L$.

Proof We prove this by induction over $\mathcal{DB}$. Constants are trivial.

$\boxed{\mathsf{var}(i)}$ $(\!|\mathsf{var}(i)|\!)_L = v_i$. If $k \geq \mathsf{Max}(\mathsf{var}(i); L) + 1$ then $k > i$ and we are done.

$\boxed{\mathsf{bnd}(j)}$ $(\!|\mathsf{bnd}(j)|\!)_L = \mathsf{el}\, j\, L$. Similar to previous case, k is strictly greater than any index in L.

$\boxed{\mathsf{abs}(D)}$ Pick $k \geq \mathsf{Max}(\mathsf{abs}(D); L) + 1 = \mathsf{Max}(D; L) + 1$. Note that

$$(\!|\mathsf{abs}(D)|\!)_L \stackrel{\text{def}}{=} \lambda\, v_{M+1}.\, (\!|D|\!)_{v_{M+1},L}$$

where $M \stackrel{\text{def}}{=} \mathsf{Max}(D; L)$. Thus $k \geq M + 1 = \mathsf{Max}(D; v_{M+1}, L)$. If $k = M + 1$ then v_k is not free in $(\!|\mathsf{abs}(D)|\!)_L$, as any free occurrence will be captured. If $k > M + 1$ then $k \geq \mathsf{Max}(D; v_{M+1}, L) + 1$ and so by induction v_k is not free in $(\!|D|\!)_{v_{M+1},L}$ and so we are done.

$\boxed{D_1\ \$\ D_2}$ This case is easy and omitted.

□

Lemma A.8 Let L', L and $\hat{L}, L$ be ordered lists, with $|L'| = |\hat{L}| \geq 1$. Let $D \in \mathcal{DB}(|\hat{L}, L|)$. Then

$$(\!|D|\!)_{\hat{L},L}[L'/\hat{L}] \sim_\alpha (\!|D|\!)_{L',L}$$

whenever

$$\mathsf{Min}\{k \mid \exists v_k \in L'\} \geq \mathsf{Max}(D; \hat{L}, L) + \#\mathsf{Abs}(D) + 1 \qquad (*)$$

where $\#\mathsf{Abs}(D)$ is the number of "abstraction" nodes in D; and

$$\mathsf{Min}\{k \mid \exists v_k \in \hat{L}\} \geq \mathsf{Max}(D; \epsilon) + 1 \qquad (**)$$

Proof

The proof is by induction over $\mathcal{DB}$. Constants are trivial.

$\boxed{\mathsf{bnd}(j)}$ We have

$$(\!|\mathsf{bnd}(j)|\!)_{\hat{L},L}[L'/\hat{L}] \stackrel{\text{def}}{=} (\mathsf{el}\, j\, (\hat{L}, L))[L'/\hat{L}] = \mathsf{el}\, j\, (L', L) \stackrel{\text{def}}{=} (\!|\mathsf{bnd}(j)|\!)_{L',L}$$

where each step follows from the definitions, and the second equality holds because L', L and $\hat{L}, L$ are ordered and $|\hat{L}| = |L'|$.

$\boxed{\mathsf{var}(i)}$

$$(\!|\mathsf{var}(i)|\!)_{\hat{L},L}[L'/\hat{L}] \stackrel{\text{def}}{=} v_i[L'/\hat{L}] = v_i \stackrel{\text{def}}{=} (\!|\mathsf{var}(i)|\!)_{L',L}$$

where the second equality holds because any index in $\hat{L}$ is greater than or equal to $\mathsf{Max}(\mathsf{var}(i);\epsilon)+1 = i+1 > i$ by assumption $(**)$.

$\boxed{\mathsf{abs}(D)}$ Suppose that $M \stackrel{\text{def}}{=} \mathsf{Max}(\mathsf{abs}(D);\hat{L},L)$ and

$$\begin{aligned}\mathsf{Min}\{k \mid \exists v_k \in L'\} &\geq \mathsf{Max}(\mathsf{abs}(D);\hat{L},L) + \#\mathsf{Abs}(\mathsf{abs}(D)) + 1 \\ &= M + (\#\mathsf{Abs}(D)+1) + 1 \qquad (i) \\ &> M+1 \qquad (ii)\end{aligned}$$

and $\mathsf{Min}\{k \mid \exists v_k \in \hat{L}\} \geq \mathsf{Max}(\mathsf{abs}(D);\epsilon)+1 = \mathsf{Max}(D;\epsilon)+1 \quad (iii)$

Then we have

$$\begin{aligned}(\!|\mathsf{abs}(D)|\!)_{\hat{L},L}[L'/\hat{L}] &\stackrel{\text{def}}{=} (\lambda\, v_{M+1}.\,(\!|D|\!)_{v_{M+1},\hat{L},L})[L'/\hat{L}] && \text{(A.1)}\\ &= \lambda\, v_{M+1}.\,(\!|D|\!)_{v_{M+1},\hat{L},L}[L'/\hat{L}] && \text{(A.2)}\\ &\sim_\alpha \lambda\, v_{M'+1}.\,(\!|D|\!)_{v_{M+1},\hat{L},L}[L'/\hat{L}][v_{M'+1}/v_{M+1}] && \text{(A.3)}\\ &\sim_\alpha \lambda\, v_{M'+1}.\,(\!|D|\!)_{v_{M+1},\hat{L},L}[L', v_{M'+1}/\hat{L}, v_{M+1}] && \text{(A.4)}\\ &= \lambda\, v_{M'+1}.\,(\!|D|\!)_{v_{M+1},\hat{L},L}[v_{M'+1}, L'/v_{M+1}, \hat{L}] && \text{(A.5)}\\ &\sim_\alpha \lambda\, v_{M'+1}.\,(\!|D|\!)_{v_{M'+1},L',L} && \text{(A.6)}\\ &\stackrel{\text{def}}{=} (\!|\mathsf{abs}(D)|\!)_{L',L} && \text{(A.7)}\end{aligned}$$

From (ii) $v_{M+1} \notin L'$ and so the substitution in equation (A.1) does not involve re-naming. Further, recall that by definition, $M \stackrel{\text{def}}{=} \mathsf{Max}(\mathsf{abs}(D);\hat{L},L)$. Hence $v_{M+1} \notin \hat{L}$ and so, recalling the definition of substitution, equation (A.2) holds with $\overline{L'} = L'$ and $\overline{\hat{L}} = \hat{L}$.

We set $M' \stackrel{\text{def}}{=} \mathsf{Max}(D;L',L)$ and so $M' > M+1$ by (ii). By appeal to Lemma A.7, $v_{M'+1}$ is not free in $(\!|D|\!)_{v_{M+1},\hat{L},L}$ provided that

$$M'+1 \geq \mathsf{Max}(D;v_{M+1},\hat{L},L)+1.$$

But this holds, as $M' > M+1$, and by (iii) and list order, we have

$$\mathsf{Max}(D; v_{M+1}, \hat{L}, L) = M+1$$

Further, $M'+1$ is strictly greater than the indices in L' by definition, and thus $v_{M'+1}$ is not free in $(\!|D|\!)_{v_{M+1},\hat{L},L}[L'/\hat{L}]$. By the axiom for α-equivalence, equation (A.3) holds.

Again, as $v_{M+1} \notin L'$ (proved above), by Lemma A.4 we have equation (A.4).

The equality (A.5) holds as we have $v_{M+1} \notin \hat{L}$ (proved above).

The equality (A.6) holds by induction together with the congruence of abstraction, as the conditions of the lemma both hold as follows: Note that $(*)$ is true as

$$\begin{aligned}\mathsf{Min}\{k \mid \exists v_k \in v_{M'+1}, L'\} &= \mathsf{Min}\{k \mid \exists v_k \in L'\} \\ &\geq (M+1) + (\#\mathsf{Abs}(D) + 1) \\ &= \mathsf{Max}(D; v_{M+1}, \hat{L}, L) + \#\mathsf{Abs}(D) + 1\end{aligned}$$

where the inequality holds by (i) and final equality follows from the arguments above. Further, $(**)$ is true because (iii) implies

$$\mathsf{Min}\{k \mid \exists v_k \in v_{M+1}, \hat{L}\} = \mathsf{Min}\{k \mid \exists v_k \in \hat{L}\} \geq \mathsf{Max}(D; \epsilon) + 1$$

$D_1 \,\$\, D_2$ The easy details are omitted.

□

Corollary A.9 For any ordered L, and $D \in \mathcal{DB}$, there is a sufficiently large $w \geq 0$ for which

$$(\!|\mathsf{abs}(D)|\!)_L \sim_\alpha \lambda\, v_w.\, (\!|D|\!)_{v_w,L}$$

Proof Recall that $(\!|\mathsf{abs}(D)|\!)_L \stackrel{\text{def}}{=} \lambda\, v_{M+1}.\, (\!|D|\!)_{v_{M+1},L}$ with $M \stackrel{\text{def}}{=} \mathsf{Max}(D; L)$. In particular v_{M+1}, L is ordered. It follows from Lemma A.8 that

$$(\!|D|\!)_{v_{M+1},L}[v_w/v_{M+1}] \sim_\alpha (\!|D|\!)_{v_w,L} \qquad \dagger$$

provided that $(*)$ and $(**)$ hold. $(*)$ holds provided we choose

$$w \geq \mathsf{Max}(D; v_{M+1}, L) + \#\mathsf{Abs}(D) + 1$$

(**) holds because

$$\mathsf{Min}\{k \mid v_k \in v_{M+1}\} = M + 1 = \mathsf{Max}(D; L) + 1 \geq \mathsf{Max}(D; \epsilon) + 1$$

Further, v_w, L is ordered. We have

$$\lambda\, v_w.\, (\!|D|\!)_{v_w, L} \sim_\alpha \lambda\, v_w.\, (\!|D|\!)_{v_{M+1}, L}[v_w/v_{M+1}]$$

by applying a congruence rule to †; and

$$\lambda\, v_w.\, (\!|D|\!)_{v_{M+1}, L}[v_w/v_{M+1}] \sim_\alpha \lambda\, v_{M+1}.\, (\!|D|\!)_{v_{M+1}, L} = (\!|\mathsf{abs}(D)|\!)_L$$

by appeal to Lemma A.7, noting $w \geq \mathsf{Max}(D; v_{M+1}, L) + 1$ implies $v_w \notin fv(\!|D|\!)_{v_{M+1}, L}$, along with an instance of the axiom for α-equivalence. □

Proposition A.10 Let $E \in \mathcal{LE}$, and let L and L' be lists, with L' ordered, such that $|L| = |L'|$. Then

$$(\!|[\![E]\!]_L|\!)_{L'} \sim_\alpha E[L'/L]$$

Proof

We apply induction over $\mathcal{LE}$. As ever, constants are trivial.

$\boxed{v_i}$

If $v_i \notin L$, then $(\!|[\![v_i]\!]_L|\!)_{L'} \stackrel{\text{def}}{=} (\!|\mathsf{var}(i)|\!)_{L'} \stackrel{\text{def}}{=} v_i = v_i[L'/L]$.

If $v_i \in L$, then

$$(\!|[\![v_i]\!]_L|\!)_{L'} \stackrel{\text{def}}{=} (\!|\mathsf{pos}\, v_i\, L|\!)_{L'} \stackrel{\text{def}}{=} \mathsf{el}\, (\mathsf{pos}\, v_i\, L)\, L' \stackrel{\text{def}}{=} v_i[L'/L]$$

$\boxed{E_1\ E_2}$ The details are routine.

$\boxed{\lambda\, v_i.\, E}$

$$\begin{aligned}
(\!|[\![\lambda\, v_i.\, E]\!]_L|\!)_{L'} &\stackrel{\text{def}}{=} (\!|\mathsf{abs}([\![E]\!]_{v_i, L})|\!)_{L'} && \text{(A.8)}\\
&\sim_\alpha \lambda\, v_w.\, (\!|[\![E]\!]_{v_i, L}|\!)_{v_w, L'} && \text{(A.9)}\\
&\sim_\alpha \lambda\, v_w.\, E[v_w, L'/v_i, L] && \text{(A.10)}\\
&\sim_\alpha \lambda\, v_w.\, E[v_w/v_i][L'/L] && \text{(A.11)}
\end{aligned}$$

$$= (\lambda\, v_w.\, E[v_w/v_i])[L'/L] \tag{A.12}$$
$$\sim_\alpha (\lambda\, v_i.\, E)[L'/L] \tag{A.13}$$

Equivalence (A.9) holds by appeal to Corollary A.9 where w is also chosen large enough to be fresh for E, L, L' and v_i. Equivalence (A.10) holds by induction, noting that v_w, L' is indeed ordered. Equivalence (A.11) holds by appeal to Lemma A.4. Equation (A.12) follows from the definition of substitution; note that the choice of w ensures that there is no deletion of mate pairs. The final step (A.13) follows using the axiom of α-equivalence, and Proposition A.2.

□

We now show that the function $[\![-]\!]_\epsilon : \mathcal{LE} \to \mathcal{PDB}$ is equal on α-equivalent expressions, using the following lemmas:

Lemma A.11 For any $E \in \mathcal{LE}$ and lists L and L', and any v_k, if the following conditions

$$v_k \notin fv(E) \vee v_k \in L' \qquad (*)$$

and

$$v_{k'} \notin fv(E) \vee v_{k'} \in L' \qquad (**)$$

hold, then

$$[\![E]\!]_{L',v_k,L} = [\![E]\!]_{L',v_{k'},L}$$

Proof We use induction over $\mathcal{LE}$. Constants are trivial.

$\boxed{v_i}$ We have to check that

$$[\![v_i]\!]_{L',v_k,L} = [\![v_i]\!]_{L',v_{k'},L}$$

This requires a case analysis. If $v_i \in L'$ we are done. Now suppose $v_i \notin L'$. Note that if $v_i = v_k$ then by condition $(*)$ we must have $v_i \notin v_i$, a contradiction. Thus $v_i \neq v_k$. A symmetric argument for $(**)$ shows that $v_i \neq v_{k'}$. Thus either $v_i \in L$ and we are done, or in fact v_i is not in *any* of the lists and both sides of the required equality are equal to $\mathsf{var}(i)$.

$\boxed{\lambda\, v_i.\, E}$ We have to check that

$$\mathsf{abs}([\![E]\!]_{v_i,L',v_k,L}) = \mathsf{abs}([\![E]\!]_{v_i,L',v_{k'},L})$$

under the assumptions

$$v_k \notin fv(\lambda\, v_i.\, E) \vee v_k \in L'$$

and

$$v_{k'} \notin fv(\lambda\, v_i.\, E) \vee v_{k'} \in L'$$

The equality will follow by induction provided that both

$$v_k \notin fv(E) \vee v_k \in v_i, L' \qquad (*)$$

and

$$v_{k'} \notin fv(E) \vee v_{k'} \in v_i, L' \qquad (**)$$

hold. In $(*)$ suppose that $v_k \notin v_i, L'$. We must then have $v_k \notin fv(\lambda\, v_i.\, E)$ and $v_k \neq v_i$, so $v_k \notin fv(E)$. Thus $(*)$ holds. The argument for $(**)$ is analogous.

$\boxed{E_1\ E_2}$ The easy details are omitted.

□

Lemma A.12 Let $E \in \mathcal{LE}$, let L' and L be any lists, and let v_k, $v_{k'}$ be variables for which $v_{k'} \notin fv(E)$, $v_k \notin L'$ and $v_{k'} \notin L'$. Then we have

$$[\![E[v_{k'}/v_k]]\!]_{L', v_{k'}, L} = [\![E]\!]_{L', v_k, L} \qquad (*)$$

Proof We prove this result by induction on the size of the expression E, where constants and variables have size 1, the size of an application is the sum of the sizes of the two subterms, and the size of an abstraction is the size of the body plus 1. Write $\mathsf{size}(E)$ for the size of E and $\Phi(E)$ for $(*)$ in which L, L', k and k' are universally quantified and satisfy the given constraints. Write $\Psi(n)$ for

$$(\forall E)(\mathsf{size}(E) = n \Longrightarrow \Phi(E))$$

and we prove $\forall n.\Phi(n)$ by strong induction on n. Note (carefully!) that we have to prove $\Psi(1)$ explicitly—the base case for the induction. We write *LHS* and *RHS* for the left and right hand sides of the equality in the lemma.

$\boxed{\Psi(1)}$ Consider arbitrary expressions of size 1. If a constant the result is immediate. Otherwise we have a variable, say v_i. We have to prove $\Phi(v_i)$. Note that $v_{k'}$ must not be free in v_i, so $i \neq k'$.

(Case $i = k$): If $i = k$ then

$$LHS = [\![v_{k'}]\!]_{L',v_{k'},L} = \mathsf{pos}\, v_{k'}\, (L', v_{k'}, L) = \\ \mathsf{pos}\, v_i\, (L', v_k, L) = [\![v_i]\!]_{L',v_k,L} = RHS$$

where the positions are equal due to the constraints of the lemma concerning L'.

(Case $i \neq k$): We have to prove that $[\![v_i]\!]_{L',v_{k'},L} = [\![v_i]\!]_{L',v_k,L}$. If $v_i \in L'$ we are done. Now suppose $v_i \notin L'$. Note further that $k \neq i \neq k'$. Hence either $v_i \in L$ and we are done, or else $v_i \notin L$ and then $\mathsf{var}(i) \stackrel{\text{def}}{=} [\![v_i]\!]_{L',v_{k'},L} = [\![v_i]\!]_{L',v_k,L} \stackrel{\text{def}}{=} \mathsf{var}(i)$.

$\boxed{(\forall n)((\forall m < n)(\Psi(m) \Longrightarrow \Psi(n)))}$ Choose arbitrary $n \geq 2$, and suppose that $\Psi(m)$ holds for all m smaller than n. We must prove $\Psi(n)$. So consider an arbitrary expression N of size n. We have to prove that $\Phi(N)$ holds, assuming $\Phi(M)$ for all expressions M of size smaller than N.

In the case when $N \in \mathcal{LE}$ is of the form $E_1\ E_2$ the result follows by a routine inductive argument which we omit.

In the case when $N \in \mathcal{LE}$ is of the form $\lambda\, v_i.\, E$, note that we can assume that $\Phi(M)$ for any M with $\mathsf{size}(M) < \mathsf{size}(E) + 1$. We have to prove that

$$LHS \stackrel{\text{def}}{=} [\![(\lambda\, v_i.\, E)[v_{k'}/v_k]]\!]_{L',v_{k'},L} = [\![\lambda\, v_i.\, E]\!]_{L',v_k,L} \stackrel{\text{def}}{=} RHS$$

when $v_{k'} \notin fv(\lambda\, v_i.\, E) \stackrel{\text{def}}{=} fv(E) \setminus \{v_i\}$, and $v_k \notin L'$ and $v_{k'} \notin L'$.

(Case $i = k$): Here the variable v_k is not free in $\lambda\, v_i.\, E$ and so $LHS = \mathsf{abs}([\![E]\!]_{v_i,L',v_{k'},L})$ and further $RHS = \mathsf{abs}([\![E]\!]_{v_i,L',v_k,L})$. Equality follows from Lemma A.11—we check the assumptions of the lemma are satisfied. Note that if $i = k'$ then we are (trivially) done. So suppose $i \neq k'$. $(*)$ holds as $i = k$ implies $v_k \in v_i, L$. $(**)$ holds as $v_{k'} \notin fv(E) \setminus \{v_i\}$ and $i \neq k'$ imply $v_{k'} \notin fv(E)$.

(Case $i \neq k$): Here the variable v_k may be free in $\lambda\, v_i.\, E$.

(Subcase $v_k \notin fv(E)$ or $v_i \neq v_{k'}$): This case is when the substitution does not involve a renaming. Note that $LHS = \mathsf{abs}([\![E[v_{k'}/v_k]]\!]_{v_i,L',v_{k'},L})$ and $RHS = \mathsf{abs}([\![E]\!]_{v_i,L',v_k,L})$.

In the case that $v_i \neq v_{k'}$ then $v_{k'} \notin fv(E)$ follows using the same reasoning as in case $i = k$ above. Further, as $i \neq k$ and $i \neq k'$, $LHS = RHS$ follows inductively from $\Phi(E)$, as $\mathsf{size}(E) < \mathsf{size}(E) + 1$.

In the case that $v_k \notin fv(E)$ (and $i = k'$ say), then $LHS = \mathsf{abs}(\llbracket E \rrbracket_{v_i, L', v_{k'}, L})$. Equality follows from Lemma A.11—we check the assumptions of the lemma are satisfied. $(*)$ holds as $v_k \notin fv(E)$. $(**)$ holds as $i = k'$ implies $v_{k'} \in v_i, L$.

(Subcase $v_k \in fv(E)$ and $v_i = v_{k'}$): Here we have

$$
\begin{aligned}
LHS &= \llbracket \lambda\, v_w.\, E[v_w/v_i][v_{k'}/v_k] \rrbracket_{L', v_{k'}, L} && \text{(A.14)} \\
&\stackrel{\text{def}}{=} \mathsf{abs}(\llbracket E[v_w/v_i][v_{k'}/v_k] \rrbracket_{v_w, L', v_{k'}, L}) && \text{(A.15)} \\
&= \mathsf{abs}(\llbracket E[v_w/v_i] \rrbracket_{v_w, L', v_k, L}) && \text{(A.16)} \\
&= \mathsf{abs}(\llbracket E \rrbracket_{v_i, L', v_k, L}) && \text{(A.17)} \\
&\stackrel{\text{def}}{=} RHS && \text{(A.18)}
\end{aligned}
$$

In equation (A.14), w is the maximum of the indices in E, v_k and $v_{k'}$, plus 1. Equation (A.16) follows from inductive hypothesis $\Phi(E[v_w/v_i])$, since

$$\mathsf{size}(E[v_w/v_i]) = \mathsf{size}(E) < \mathsf{size}(E) + 1,$$

both v_k and $v_{k'}$ are not in v_w, L', and also $v_{k'} \notin fv(E[v_w/v_i])$ using the original assumption about $v_{k'}$ and the definition of w. Equation (A.17) follows by induction in a similar fashion. This completes the proof.

□

Proposition A.13 For any L, if $E \sim_\alpha E'$ then $\llbracket E \rrbracket_L = \llbracket E' \rrbracket_L$. In particular $\llbracket E \rrbracket_\epsilon = \llbracket E' \rrbracket_\epsilon$.

Proof By induction on the axioms and rules defining alpha equivalence, one proves

$$(\forall (E, E') \in \sim_\alpha)(\forall L)(\llbracket E \rrbracket_L = \llbracket E' \rrbracket_L)$$

The only difficult part concerns the axiom $\lambda\, v_k.\, E \sim_\alpha \lambda\, v_{k'}.\, E[v_{k'}/v_k]$ in which $v_{k'}$ is chosen so that it is not free in E. We have

$$\llbracket \lambda\, v_k.\, E \rrbracket_L \stackrel{\text{def}}{=} \mathsf{abs}(\llbracket E \rrbracket_{v_k, L}) = \mathsf{abs}(\llbracket E[v_{k'}/v_k] \rrbracket_{v_{k'}, L}) \stackrel{\text{def}}{=} \llbracket \lambda\, v_k.\, E[v_{k'}/v_k] \rrbracket_L$$

The equality follows by appealing to Lemma A.12, with $L' = \epsilon$ so that (trivially) v_k and $v_{k'}$ are not in L'.

□

We can now prove Theorem 3.1.

Proof Consider the following diagram, with q the surjective quotient map.

$$\theta : \mathcal{LE}/{\sim_\alpha} \xleftarrow{\quad q \quad} \mathcal{LE} \underset{(\!|-|\!)_\epsilon}{\overset{[\![-]\!]_\epsilon}{\rightleftarrows}} \mathcal{PDB} : \phi$$

We define $\theta([E]_\alpha) \stackrel{\text{def}}{=} [\![E]\!]_\epsilon$ for any $E \in \mathcal{LE}$ and $\phi \stackrel{\text{def}}{=} q \circ (\!|-|\!)_\epsilon$. Note that the definition of θ is a good one, by appealing to Proposition A.13 which shows that $[\![-]\!]_\epsilon$ is equal on α-equivalent expressions. We then have

$$(\theta \circ \phi)(D) = [\![(\!|D|\!)_\epsilon]\!]_\epsilon = D$$

using Proposition A.1, and

$$(\phi \circ \theta)[E]_\alpha = [(\!|[\![E]\!]_\epsilon|\!)_\epsilon]_\alpha = [E]_\alpha$$

using Proposition A.10. □

References

[1] Penny Anderson and Frank Pfenning Verifying Uniqueness in a Logical Framework In V. A. Carreño, editor, *Proceedings of the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04)*, volume 3223 of *LNCS*. Springer Verlag, 2004.

[2] Simon Ambler, Roy Crole, and Alberto Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In V. A. Carreño, editor, *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002*, volume 2342 of *LNCS*. Springer Verlag, 2002.

[3] S. J. Ambler and R. L. Crole and A. Momigliano. A Combinator and Presheaf Topos Model for Primitive Recursion over Higher Order Abstract Syntax. In Collegium Logicum (Proceedings of the Kurt Godel Society), pages 83-89, Springer-Verlag, 2004. ISBN: 3-901546-03-0

[4] 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.

[5] N. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

[6] 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.

[7] J. Despeyroux and P. Leleu. Metatheoretic results for a modal λ-calculus. *Journal of Functional and Logic Programming*, 2000(1), 2000.

[8] J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, Apr. 1997. Springer-Verlag LNCS.

[9] M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214–224, Trento, Italy, 1999. IEEE Computer Society Press.

[10] A. D. Gordon and T. Melham. Five axioms of alpha-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–190, Turku, Finland, August 1996. Springer-Verlag.

[11] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, Jan. 1993.

[12] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *Transactions on Computational Logic*, 6:61–101, January 2005.

[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] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transaction in Computational Logic*, 2001. To appear.

[16] D. Miller. Representing and reasoning with operational semantics. *Third International Joint Conference on Automated Reasoning, 17-20 August, 2006, Seattle.* Springer Verlag.

[17] J. McKinna and R. Pollack. Some Type Theory and Lambda Calculus Formalised. To appear in Journal of Automated Reasoning, Special Issue on Formalised Mathematical Theories (F. Pfenning, Ed.),

[18] T. F. Melham. A mechanized theory of the π-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, Spring 1994.

[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] 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.

[21] F. Pfenning. Computation and deduction. Lecture notes, 277 pp. Revised 1994, 1996, to be published by Cambridge University Press.

[22] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1988.

[23] A. M. Pitts. Nominal logic: A first order theory of names and binding. In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29-31, 2001, Proceedings*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–242. Springer-Verlag, Berlin, 2001.

[24] A. M. Pitts. Alpha-Structural Recursion and Induction (Extended Abstract) In J. Hurd and T. Melham, editor, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford UK, August 2005*, volume 3603, pages 17–34 of *LNCS*. Springer-Verlag, 2005.

[25] A. M. Pitts. Alpha-Structural Recursion and Induction *Journal of the ACM*, 53:459–506, 2006.

[26] R. Vestergaard and J. Brotherson. A formalized first-order conflence proof for the λ-calculus using one sorted variable names. In A. Middelrop, editor, *Proceedings of RTA '12*, volume 2051 of *LNCS*, pages 306–321. Springer-Verlag, 2001.