A Definitional Approach to Primitive Recursion over Higher Order Abstract Syntax ⋆

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

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

Abstract. It is well known that there are problems associated with formal systems which attempt to combine higher order abstract syntax (HOAS) with principles of induction and recursion. We describe a formal system, called Bsyntax, which we have implemented in Isabelle HOL. Our contribution is to prove the existence of a combinator for primitive recursion with parameters over HOAS. The definition of the combinator is facilitated by the use of terms with *infinite* contexts. In particular, our work is purely definitional, and is thus consistent with classical logic and choice. An immediate payoff is that we obtain a primitive recursive definition of higher order substitution. We give a presheaf model of Bsyntax, providing additional semantic validation of Bsyntax's principles of recursion. We outline an application of our work to mechanized reasoning about the compiler intermediate language MIL-lite [2].

1 Introduction

Higher order abstract syntax (HOAS) has been the subject of considerable research efforts over the last few years. The fundamental idea, dating back to [4], is to represent the variables of an object level logic by the variables of a meta-logic (and terms of the meta-logic represent terms of the object level logic). Thus, in particular, variable binding in the object logic is represented by variable binding in the meta-logic, and functions such as substitution can be defined once and for all in the meta-logic. In this paper, the "meta-logic" we use as a basis for HOAS is Isabelle HOL.

It is well known that various problems can arise when trying to combine HOAS with principles of induction, see, for example, [13, 27, 1]. One particular problem concerns how to define recursive functions over the terms of HOAS. In order to state properties of and reason about object logics, we may want to employ definitions by (primitive) recursion. For example, to encode the operational semantics of a (object level) functional programming language we require meta-level capture avoiding substitution, which can be defined by primitive recursion (with parameters). Of course, functions such as substitution can be defined in other ways, often as a relation which one attempts to prove total and functional. But such approaches are often quite messy in practice. The key issue here is how to define recursive calls over terms involving λ-binders. This is problematic, and is discussed in detail in [27, 26]. The main contributions of this paper are

– a presentation of weak HOAS [22] using a λ-calculus of *terms with infinite contexts*, coded in Isabelle HOL;

⋆ This work was supported by EPSRC grant number GR/M98555.
⋆⋆⋆ Corresponding author. Research supported by leave from the University of Leicester.

– a proof of existence of a *combinator for primitive recursion over HOAS*, coded in Isabelle HOL;
– a *presheaf topos model*, from which we obtain semantic validation of recursion principles by exhibiting the types over which recursion takes place as initial algebras;
– an outline of the representation of a *substantial object logic*, namely the compiler intermediate language MIL-lite [2], together with machine proofs of properties of the system.

We refer to our Isabelle HOL Theories by the name `Bsyntax` (binding syntax).

In Section 2 we introduce a datatype for HOAS and show how to identify a subtype of λ-*calculus terms-in-context*. In Section 3 we motivate and introduce a combinator for primitive recursion. In Section 4 we show how to capture the semantics of a very simple programming language within `Bsyntax`, and how to define a family of higher order substitution functions using the combinator. We outline the MIL-lite language of Benton and Kennedy [2] and show that we can also capture its semantic description, and prove program properties. In Section 5 we present an abstract presheaf topos model of our representation of λ-calculus. In Sections 6 and 7 we review related work and draw some conclusions. The Appendix A contains proofs of selected results.

2 An Encoding of λ-calculus Terms-in-Infinite-Contexts

We define the type $var \stackrel{\text{def}}{=} nat$. The datatype below will be used to give a general exposition of our definition of expressions of (weak [22]) HOAS; it is implemented in Isabelle HOL as part of `Bsyntax`.

$$\mathsf{datatype} \quad exp \quad ::= \quad \mathsf{V}\ var \mid \mathsf{Lambda}\ (var \Rightarrow exp) \mid exp\ \mathsf{A}\ exp$$

Note that the actual datatype which one sees in our Isabelle HOL implementation has a clause $\mathsf{C}\ string$ specifying constants, and a clause ERR specifying an "error" term. Constants are crucial in a logical framework for naming the constructors of object logics. The error type is used to take care of exotic terms [6]. While an important part of `Bsyntax`, for pedagogical reasons we have chosen to suppress these technical aspects.

A typical abstraction term $\mathsf{Lambda}\ (\lambda\, v.\, e\ v)$ of type *exp* will be written using the sugared notation $\mathsf{L}\ v.\ e\ v$. From this datatype we wish to extract those terms of type *exp* which correspond to terms of the λ-calculus. To do this, first recall the standard encoding of an object level λ-calculus into terms of type *exp*. If the λ-calculus has a set Λ of terms given by

$$E ::= V \mid E\ E' \mid \Lambda V.E$$

then we can define a translation into our datatype by setting

$$\begin{aligned} \ulcorner V \urcorner &\stackrel{\text{def}}{=} \mathsf{V}\ v \\ \ulcorner E\ E' \urcorner &\stackrel{\text{def}}{=} \ulcorner E \urcorner\ \mathsf{A}\ \ulcorner E' \urcorner \\ \ulcorner \Lambda V.E \urcorner &\stackrel{\text{def}}{=} \mathsf{L}\ v.\ \ulcorner E \urcorner \end{aligned}$$

As is well known [6], the function $\ulcorner - \urcorner : \Lambda \to exp$ is not a bijection, due to the presence of exotic terms. Moreover, in order to obtain a representation of the λ-calculus in which the usual operational properties are correctly modeled, the translation function should also preserve substitutions. Any such translation function is said to provide an *adequate* representation. We obtain such a function by carving out a subtype *lam* of so called "proper" terms of type *exp* such that the function $\ulcorner - \urcorner : \Lambda \to lam$ is indeed a compositional bijection. We aim to define a function

$$\frac{}{\mathsf{prop}\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ n))} \qquad \frac{\mathsf{prop}\ e_1 \qquad \mathsf{prop}\ e_2}{\mathsf{prop}\ (\lambda\, l.\ (e_1\ l)\ \mathsf{A}\ (e_2\ l))} \qquad \frac{\mathsf{prop}\ (\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l))}{\mathsf{prop}\ (\lambda\, l.\ \mathsf{L}\ u.\ e\ u\ l)}$$

Table 1. Definition of proper terms-in-infinite-contexts

$\mathsf{prop} :: \mathit{exp} \Rightarrow \mathit{bool}$ such that $\mathit{lam} = \{e \mid e :: \mathit{exp} \wedge \mathsf{prop}\ e\}$. Let us see what happens if we proceed naively by recursion. We might define

$$\frac{}{\mathsf{prop}\ (\mathsf{V}\ v)} \qquad \frac{\mathsf{prop}\ e_1 \quad \mathsf{prop}\ e_2}{\mathsf{prop}\ (e_1\ \mathsf{A}\ e_2)} \qquad \frac{?}{\mathsf{prop}\ (\mathsf{L}\ v.\ e\ v)}$$

but then one has to consider a problem which arises in the final clause. The type of e is $\mathit{var} \Rightarrow \mathit{exp}$, and not exp which prop expects. We could try defining the antecedent as $\forall v.\ \mathsf{prop}\ (e\ (\mathsf{V}\ v))$ so that binders are traversed via a meta-level universal quantification. This can be used with some success [1]. However, there can be serious drawbacks when reasoning on the left of meta-logical sequents [19, 20], typically when performing an induction over open terms, but also when performing simple inversions. We could also try $\forall x.e\ \mathsf{prop}\ x \longrightarrow \mathsf{prop}\ (e\ x)$, but this leads to a non-monotonic definition which is rejected by a traditional proof assistant based on inductive definitions (or types). In this paper we develop another approach. By thinking about the way in which the L binder interacts with free and bound variables, one is lead to consider defining judgments over terms-in-context, in particular here prop. Traditionally, a term-in-context takes the form $\Gamma \vdash e$, where Γ is a finite list of variables occurring free in e. Now, such terms-in-context are usually identified up to a consistent renaming of the variables which occur in the context. As such, we can capture the notion by regarding the context as a variable binding operation at the Isabelle HOL meta-level. A term-in-context would take the form $\lambda\, v_0 \ldots v_{r-1}.\ e\ v_0\ \ldots\ v_{r-1} :: \mathit{var}^r \Rightarrow \mathit{exp}$. We would then define functions $\mathsf{prop}\ r :: \mathit{var}^r \Rightarrow \mathit{exp}$ for any $r \geq 0$ which satisfy

$$\frac{\mathsf{prop}\ (r+1)\ (\lambda\, v_0 \ldots v_r.\ e\ v_0 \ldots v_r)}{\mathsf{prop}\ r\ (\lambda\, v_0 \ldots v_{r-1}.\ \mathsf{L}\ u.\ e\ v_0\ \ldots\ v_{r-1}\ u)}$$

This approach will also not work in Isabelle HOL, which does not have dependent types.

We circumvent such problems by using a method which is founded on similar approaches in [7, 14]. The key idea is to *work with terms-in-context* as motivated above where the contexts are *infinite*. In particular, a context will be a *stream* of variables, realized as a term of type $\mathit{var\ stream} \stackrel{\mathrm{def}}{=} \mathit{nat} \Rightarrow \mathit{var}$[1]. One reason for working with infinite contexts is that some of the bookkeeping tasks mentioned above are simplified. In particular, we make use of the functions which compute the nth element of, the tail of, and drop n elements from a list l, denoted by $(l\ !\ n)$, $\mathsf{tl}\ l$ and $\mathsf{dr}^n\ l$, while u consed onto l is denoted by $u\ \#\ l$. Note that over finite lists, tail and drop are not total, which can complicate matters in a theorem prover such as Isabelle HOL which disallows partial functions. Each such term-in-infinite-context has type $\mathit{eic} \stackrel{\mathrm{def}}{=} \mathit{var\ stream} \Rightarrow \mathit{exp}$; a typical one has the η-long form $\lambda\, l.\ e\ l :: \mathit{eic}$. The revised definition of $\mathsf{prop} :: \mathit{eic} \Rightarrow \mathit{bool}$ is given in Table 1.

Here is an example of a proper term.

$$\lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 4))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 8))$$

[1] This because of the lack of co-datatypes in Isabelle HOL.

This is an encoding of a λ-calculus term $\Gamma \vdash \Lambda U.U\ V_4\ V_8$, where, for example, $\Gamma(4) = V_4$. One has to take care in understanding the meaning of say $\lambda\, l.\ \mathsf{V}\ (l\ !\ 4)$. Recall that $l :: \mathit{var\ stream}$. So $\lambda\, l.\ \mathsf{V}\ (l\ !\ 4)$ is the fourth "actual" variable in a fixed enumeration. In fact we will think of it as the *fourth projection* of an arbitrary infinite sequence of variables. Note also the binder λl gives rise to a notion of context, and that traditionally contexts consist of *distinct* variables. This is indeed the case here, as we can prove that $\lambda\, l.\ \mathsf{V}\ (l\ !\ n) = \lambda\, l.\ \mathsf{V}\ (l\ !\ m)$ just in case $m = n$. We refer to $\lambda\, l.\ \mathsf{V}\ (l\ !\ n)$ as the ***n*th (variable) projection**. Moving to the definition of proper abstractions, consider for example the encoding of "properness" of $\Lambda U.U\ V_4\ V_8$:

$$\frac{U, V_1, V_2 \ldots \vdash U\ V_5\ V_9}{V_0, V_1 \ldots \vdash \Lambda U.U\ V_4\ V_8} \qquad \frac{\mathsf{prop}\ (\quad \lambda\, l.\ (\mathsf{V}\ (l\ !\ 0))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 5))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 9)) \quad)}{\mathsf{prop}\ (\quad \lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 4))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 8)) \quad)}$$

Notice that when variables are bound by the L binder, binding is forced to occur over the 0th projection. Note also that the effect of replacing $\mathsf{tl}\ l$ by l is to decrease all other projection indices by 1 when an abstraction is formed. This is a key point, and will be fundamental to achieving a definition of a recursion combinator. To help understand the formulation of the λl binder, note that the types $\mathit{var} \Rightarrow \mathit{eic}$ and eic are *isomorphic*, with $\lambda\, u.\ \lambda\, l.\ e\ u\ l$ corresponding to $\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l)$ (see Lemma 3, page 15). This isomorphism is a basic property which holds because of the definitional property of a stream of variables. Thus properness of $\lambda\, l.\ \mathsf{L}\ u.\ e\ u\ l$ occurs just in case properness of $\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l)$ occurs. We will return to this point in Section 5, when we will select specific coproducts in our presheaf model in order to correctly model this definition of abstraction—similar issues are discussed in detail in [10].

Note also that the revised definition of prop involves only Horn clauses, unlike the alternative definitions of prop alluded to above. There is no use of a meta-logical universal quantifier—the "stream binder" λl has rendered the quantification "internal" to prop—and this has payoffs when undertaking machine proofs, in particular by induction over open terms. Typically, when binding are traversed via meta-logical universal quantification, the structural induction principle tends to be too weak and has to be replaced by induction on the size of the term [16]. In this paper, many results are about the `Bsyntax` system, and have been implemented in Isabelle HOL. In such cases we indicate this as follows

Remark 1 (Isabelle HOL). The definition in Table 1 specifies a monotone operator yielding a well-defined inductive definition.

3 A Combinator for Primitive Recursion

Recall our long term aim of using some form of HOAS to *encode* object level languages with variable binding, and to *reason* about them using a proof assistant such as Isabelle HOL. In our setting, object level terms will be encoded as proper terms of type *eic*. We will often want to define functions by primitive recursion over the syntax of object level encodings. Doing this requires a measure of the size of encoded object level terms. In `Bsyntax` we can *define* such a size function through primitive recursion over the underlying proper terms of type *eic*, which is itself achieved by calling a combinator for primitive recursion. All definitions by primitive recursion are automatically functional. We will want our combinator to handle primitive recursion with parameters. In particular we can then define substitution *functionally* using the combinator. We could define substitution as a relation, and prove it total and functional; see, for example, [17]. However, such proofs of functionality may not be straightforward and must be repeated for each new function introduced by the programmer, and this is very time consuming. We believe our approach is new, and

has practical payoffs: a recursion combinator provides a single and uniform method for the direct definition of functions—all proofs that graphs of relations are total and functional are subsumed by the proof of the existence of the combinator.

To our knowledge, no-one has yet given a direct proof of the existence of such a combinator for primitive recursion over weak HOAS. The (un-curried) type of our Isabelle HOL combinator synrF is

$$(var \Rightarrow B) * (B \Rightarrow B) * (B \Rightarrow B \Rightarrow B) * eic \Rightarrow B \qquad (\dagger)$$

The type may look mysterious: it does not match up with our HOAS datatype. In order to explain the type of synr, we move to a categorical setting in which we recall the categorical analogue of such a combinator.

Let $\mathcal{C}$ be a category equipped with a strong monad (T, μ, η, τ). Think of $\mathcal{C}$ as a categorical model of `Bsyntax`. Suppose that there is a pair (Ω, σ) such that for any object P (of parameters) and morphism $f : P \times TB \to B$, there is a unique $\overline{f}$ such that

$$\begin{array}{ccc} P \times T\Omega & \xrightarrow{id \times \sigma} & P \times \Omega \\ {\scriptstyle \langle \pi_P, \mu \circ T(\eta \circ \overline{f}) \circ \tau \rangle} \downarrow & & \downarrow {\scriptstyle \overline{f}} \\ P \times TB & \xrightarrow[f]{} & B \end{array}$$

Then $\overline{f}$ is said to be defined by **recursion with parameters** over Ω. (The analogue of Ω in `Bsyntax` is of course *eic*.) Now, any category which models higher order logic must be cartesian closed. And in such a category, the following is a well known theorem, which, informally, says primitive recursion with parameters is equivalent to standard primitive recursion.

Theorem 1. *The existence of an initial algebra (Ω, σ) for the functor T of a strong monad (T, τ) over $\mathcal{C}$ is equivalent to the existence of a pair (Ω, σ) such that for any object P the square given above commutes with the given property.*

Hence we can restrict our attention to initial algebras over a cartesian closed $\mathcal{C}$. Consider the datatype for *exp* in `Bsyntax`; the defining clauses have a categorical analogue, namely a functor $T_{wh}\xi \stackrel{\text{def}}{=} var + (var \Rightarrow \xi) + \xi^2$ where *var* is an object of $\mathcal{C}$. Note that category $\mathcal{C}$ is distributive, and hence [21] the functor is a monad for which there is a strength τ. Thus Theorem 1 applies: if there exists an initial algebra (Ω, σ), then functions can be defined by primitive recursion with parameters over Ω. Any morphism $T_{wh}B \to B$ in $\mathcal{C}$ must have the form $[\mathsf{vf}, \mathsf{af}, \mathsf{lf}]$ where $\mathsf{vf} : var \to B$, $\mathsf{lf} : var \Rightarrow B \to B$ and $\mathsf{af} : B^2 \to B$, and hence there is a unique morphism $\overline{[\mathsf{vf}, \mathsf{lf}, \mathsf{af}]} : \Omega \to B$. Recall that in a (locally small) cartesian closed category, $\mathcal{C}(X, Y) \cong \mathcal{C}(1, X \Rightarrow Y)$ and $X \Rightarrow (Y \Rightarrow Z) \cong X \times Y \Rightarrow Z$. The unique morphism is thus a global element $1 \to \Omega \Rightarrow B$, and its existence is equivalent to the existence of a morphism

$$s : (var \Rightarrow B) \times ((var \Rightarrow B) \Rightarrow B) \times (B \Rightarrow B \Rightarrow B) \to \Omega \Rightarrow B$$

such that $s \circ \langle \mathsf{vf}, \mathsf{lf}, \mathsf{af} \rangle$ has the required universal property. And s corresponds to a morphism

$$1 \to (var \Rightarrow B) \times ((var \Rightarrow B) \Rightarrow B) \times (B \Rightarrow B \Rightarrow B) \Rightarrow \Omega \Rightarrow B$$

or equivalently

$$1 \to (var \Rightarrow B) \times ((var \Rightarrow B) \Rightarrow B) \times (B \Rightarrow B \Rightarrow B) \times \Omega \Rightarrow B$$

Suppose that in addition we have $var \Rightarrow \Omega \cong \Omega$. This may seem like a strange assumption—however we will see that this is indeed a property of the categorical model we will produce in Section 5. Moreover, we have already met this property in `Bsyntax`, namely $var \Rightarrow eic \cong eic$. Then equivalently we can require s to be a morphism

$$1 \to (var \Rightarrow B) \times (B \Rightarrow B) \times (B \Rightarrow B \Rightarrow B) \times \Omega \Rightarrow B$$

and the analogue of this in `Bsyntax` is precisely a combinator of type $(\dagger)$.

Having given this motivation, we can now define (the graph of) our combinator as an inductive definition in higher order logic. Note that there is a default case, omitted from this paper, when none of these patterns match.

$$\frac{}{\mathsf{synr\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ n))\ (\mathsf{vf}\ n)}$$

$$\frac{\mathsf{synr\ vf\ lf\ af}\ (\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l))\ x}{\mathsf{synr\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{L}\ u.\ e\ u\ l)\ (\mathsf{lf}\ x)} \qquad \frac{\mathsf{synr\ vf\ lf\ af}\ e_1\ y \qquad \mathsf{synr\ vf\ lf\ af}\ e_2\ z}{\mathsf{synr\ vf\ lf\ af}\ (\lambda\, l.\ (e_1\ l)\ \mathsf{A}\ (e_2\ l))\ (\mathsf{af}\ y\ z)}$$

Theorem 2 (Isabelle HOL). *The relation* synr *specified above is (the graph of) a total function* synrF*, with the required properties of a combinator for primitive recursion, namely:*

$$\begin{aligned}
\mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ n)) &= (\mathsf{vf}\ n)\\
\mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ (e_1\ l)\ \mathsf{A}\ (e_2\ l)) &= \mathsf{af}\ (\mathsf{synrF\ vf\ lf\ af}\ e_1)(\mathsf{synrF\ vf\ lf\ af}\ e_2)\\
\mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{L}\ u.\ e\ u\ l) &= \mathsf{lf}\ (\mathsf{synrF}\ \lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l))
\end{aligned}$$

This is a key result. Once again, as we saw when defining prop, we make a crucial use of the isomorphism $eic \cong var \Rightarrow eic$, which holds only because we work with the type eic of terms-in-infinite-contexts. Calling synrF over $\lambda\, l.\ \mathsf{L}\ u.\ e\ u\ l$ yields, via the isomorphism, a call over $\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l)$. The presence of the isomorphism means that the usual problem associated with recursive calls passing under binders is by-passed.

We finish this section by giving a simple example showing how synrF works in practice. Take the type B to be *nat*. Define $\mathsf{vf} \stackrel{\text{def}}{=} \lambda\, n.\ 1$ and $\mathsf{lf} \stackrel{\text{def}}{=} \lambda\, n.\ n$ and $\mathsf{af} \stackrel{\text{def}}{=} \lambda\, n.\ \lambda\, m.\ n+m$. Then synrF vf lf af will compute the number of occurrences of all variables $\mathsf{V}\ \cdot$ in a term. For example, the number N of variables in $\lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 3))$ is 2, and is given by

$$\begin{aligned}
N &= \mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 3)))\\
&= \mathsf{lf}\ (\mathsf{synrF\ vf\ lf\ af}\ ((\mathsf{V}\ (l\ !\ 0))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 4))))\\
&= \mathsf{lf}\ (\mathsf{af}\ (\mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ 0)))\ (\mathsf{synrF\ vf\ lf\ af}\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ 4))))\\
&= \mathsf{lf}\ (\mathsf{af}\ (\mathsf{vf}\ 0)\ (\mathsf{vf}\ 4)) = (\lambda\, n.\ n)\ ((\lambda\, n.\ \lambda\, m.\ n+m)\ 1\ 1) = 2
\end{aligned}$$

4 Applications to Object Level Languages

In order to illustrate how our ideas are applied in practice, we define a small (object level) language, encoding its static and dynamic semantics. While the language is elementary, we later mention that our methodology can indeed be successfully applied to a much more complex language. The types are given by integers and computation types [21]. The terms of the language are given by

$$\begin{aligned}
\mathsf{Int}\ z &\stackrel{\text{def}}{=} \mathsf{C}\ (\mathsf{stringof\ z}) & \mathsf{Val}\ e &\stackrel{\text{def}}{=} (\mathsf{C}\ \texttt{Val})\ \mathsf{A}\ e\\
e_1 + e_2 &\stackrel{\text{def}}{=} (\mathsf{C}\ \texttt{Add})\ \mathsf{A}\ e_1\ \mathsf{A}\ e_2 & \mathsf{Let}\ x \Leftarrow e_1\ \mathsf{in}\ e_2\ x &\stackrel{\text{def}}{=} (\mathsf{C}\ \texttt{Let})\ \mathsf{A}\ e_1\ \mathsf{A}\ (\mathsf{L}\ x.\ e_2\ x)
\end{aligned}$$

$$\frac{}{\Gamma \vdash \lambda\, l.\, \mathsf{V}\ (l\ !\ n) :: (\Gamma\ !\ n)} \qquad \frac{}{\Gamma \vdash \lambda\, l.\, \mathsf{Int}\ z :: \mathit{int}} \qquad \frac{\Gamma \vdash \lambda\, l.\, e\ l :: \tau}{\Gamma \vdash \lambda\, l.\, \mathsf{Val}\ (e\ l) :: \mathit{CT}\ \tau}$$

$$\frac{\Gamma \vdash \lambda\, l.\, e_1\ l :: \mathit{int} \quad \Gamma \vdash \lambda\, l.\, e_2\ l :: \mathit{int}}{\Gamma \vdash \lambda\, l.\, (e_1\ l) + (e_2\ l) :: \mathit{int}}$$

$$\frac{\Gamma \vdash \lambda\, l.\, e_1\ l :: \mathit{CT}\ \tau_1 \quad \tau_1\ \#\ \Gamma \vdash \lambda\, l.\, e_2\ (l\ !\ 0)\ (\mathsf{tl}\ l) :: \mathit{CT}\ \tau_2}{\Gamma \vdash \lambda\, l.\, \mathsf{Let}\ x \Leftarrow e_1\ l\ \mathsf{in}\ e_2\ x\ l :: \mathit{CT}\ \tau_2}$$

Table 2. A Type Assignment Relation

$$\frac{}{\lambda\, l.\, (\mathsf{Int}\ z) + (\mathsf{Int}\ z') \Downarrow \lambda\, l.\, \mathsf{Int}\ z + z'}$$

$$\frac{\lambda\, l.\, e\ l \Downarrow \lambda\, l.\, v\ l}{\lambda\, l.\, \mathsf{Val}\ (e\ l) \Downarrow \lambda\, l.\, v\ l} \qquad \frac{\lambda\, l.\, e_1\ l \Downarrow \lambda\, l.\, v_1\ l \quad \mathsf{hosub}\ e_2\ (\lambda\, l.\, v_1\ l) \Downarrow \lambda\, l.\, v\ l}{\lambda\, l.\, \mathsf{Let}\ x \Leftarrow e_1\ l\ \mathsf{in}\ e_2\ x\ l \Downarrow \lambda\, l.\, v\ l}$$

Table 3. An Evaluation Relation

Note that in this section we do make use of `Bsyntax` constants. Each constructor C has type *string* $\Rightarrow$ *exp*, and we use strings to give "names" (such as `Add`) to the constants of our object level language. Note that the let terms of a computational monad contain a binder (x above is bound) and this is captured by meta-level (`Bsyntax`) binding.

We define a ternary type assignment relation $\Gamma \vdash e :: \tau$ with carrier *types stream* $*$ *eic* $*$ *types*. The idea is that object level contexts which supply types to (free) object variables are represented by a stream of types. The relation is inductively defined using the rules in Table 2. We also define an evaluation semantics, relating terms of type *eic*, in Table 3—the substitution function hosub is explained below. In order to use `Bsyntax` to represent operational semantics in a weak HOAS setting, we must be able to represent substitution. We can define "standard" substitution via the recursion combinator as a function with the expected type $\mathsf{synrF}\ \mathsf{vf}\ \mathsf{af}\ \mathsf{lf} :: \mathit{eic} \Rightarrow \mathit{var} \Rightarrow \mathit{eic} \Rightarrow \mathit{eic}$, for suitable values of vf, lf, and af. However, in order to make proper use of HOAS, we want to be able to define higher order substitution—the recursion operator achieves this in a systematic way. In Table 3 the function hosub has type $(\mathit{var} \Rightarrow \mathit{eic}) \Rightarrow \mathit{eic} \Rightarrow \mathit{eic}$ where $e_2 :: \mathit{var} \Rightarrow \mathit{eic}$ and $\lambda\, l.\, v_1\ l :: \mathit{eic}$. In general we seek functions $\mathsf{hosub} :: (\mathit{var}^n \Rightarrow \mathit{eic}) \Rightarrow \mathit{eic}^n \Rightarrow \mathit{eic}$ for each $n \geq 1$. This we can do—again by primitive recursion—over the type $\mathit{var}^n \Rightarrow \mathit{eic}$. In a later section we shall give a categorical model which validates such a definition, by exhibiting a category with an initial algebra $T_{sr}(\mathit{var}^n \Rightarrow \mathit{eic}) \rightarrow \mathit{var}^n \Rightarrow \mathit{eic}$. Here we give a definition of hosub which is accepted by Isabelle HOL. We can define first order substitution ($n = 1$) by taking

$$\mathsf{vf} \stackrel{\text{def}}{=} \lambda\, m\ n\ f\ l.\ \mathsf{if}\ m < n\ \mathsf{then}\ \mathsf{V}\ (l\ !\ m)\ \mathsf{else}\ \mathsf{if}\ m = n\ \mathsf{then}\ f\ l\ \mathsf{else}\ \mathsf{V}\ (l\ !\ (m-1))$$

$$\mathsf{lf} \stackrel{\text{def}}{=} \lambda\, e\ n\ f\ l.\ \mathsf{L}\ u.\ e\ (\mathsf{Suc}\ n)\ (\lambda\, l.\ f\ (\mathsf{tl}\ l))\ (u\ \#\ l)$$

$$\mathsf{af} \stackrel{\text{def}}{=} \lambda\, e_1\ e_2\ n\ f\ l.\ (e_1\ n\ f\ l)\ \mathsf{A}\ (e_2\ n\ f\ l)$$

and setting

$$\mathsf{hosub} \stackrel{\text{def}}{=} \lambda\, e.\ \mathsf{synrF}\ \mathsf{vf}\ \mathsf{af}\ \mathsf{lf}\ (\lambda\, l.\ e\ (l\ !\ 0)\ (\mathsf{tl}\ l))\ 0 :: (\mathit{var} \Rightarrow \mathit{eic}) \Rightarrow \mathit{eic} \Rightarrow \mathit{eic}$$

In the general definition of lf, f is being substituted for the nth projection occurring in e. Note that in passing under a L binder, n is increased by 1, as are the projection indices in f (via tl), and the bound u is added to the context l. In the definition of af, f is being substituted for the nth projection occurring in the subterms e_1 and e_2 of an application term. In vf, f is being substituted for the nth projection; and m gives the projection at which the substitution is "currently taking place". Thus if $m = n$ then indeed f is substituted (in F below, $m = 1$ is generated from v). Indices $m < n$ are *generated* from L bound variables so remain untouched by vf (in F below, $m = 0$ is generated from L bound u). Indices $m > n$ arise from projections *already present*, so have been increased by 1 by the call of tl in hosub; hence in the case $m > n$ the indices m are reduced by 1 (in F below, $m = 5$ is generated from projection index 3).

An example may make the ideas clearer. Consider the informal substitution

$$\Lambda U.U\ V\ V_3[V := V_8] \equiv \Lambda U.U\ V_8\ V_3$$

and its formal rendition, where E is

$$E \stackrel{\text{def}}{=} \mathsf{hosub}\ (\lambda\, v.\ \lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ v)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 3)))\ (\lambda\, l.\ \mathsf{V}\ (l\ !\ 8))$$

Here, the overall effect of the function hosub should be to substitute $\lambda\, l.\ \mathsf{V}\ (l\ !\ 8)$ for the metavariable v. When hosub is called, the abstracted variable v is replaced by a 0th projection, and the call of tl ensures that all other projections are increased by 1 (4=3+1). Thus

$$E = (\underbrace{\mathsf{synrF}\ \mathsf{vf}\ \mathsf{af}\ \mathsf{lf}\ \lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ \mathsf{V}\ (l\ !\ 0)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 4)))}_{F})\ \underbrace{0}_{n}\ (\underbrace{\lambda\, l.\ \mathsf{V}\ (l\ !\ 8)}_{f})$$

Hence $F = \ldots = \mathsf{lf}\ (\mathsf{af}\ (\mathsf{af}\ (\mathsf{vf}\ \underbrace{0}_{m})\ (\mathsf{vf}\ \underbrace{1}_{m}))\ (\mathsf{vf}\ \underbrace{5}_{m}))$ and so F equals

$$\begin{array}{ll}
\lambda\, n\ f\ l\ . & \mathsf{L}\ u. \\
\quad \mathsf{A} & (\mathsf{if}\ 0 < \mathsf{Suc}\ n\ \mathsf{then}\ \mathsf{V}\ (u\ \#\ l\ !\ 0)\ \mathsf{else}\ \mathsf{if}\ 0 = \mathsf{Suc}\ n\ \mathsf{then}\ f\, l\ \mathsf{else}\ \mathsf{V}\ (u\ \#\ l\ !\ -1)) \\
\quad \mathsf{A} & (\mathsf{if}\ 1 < \mathsf{Suc}\ n\ \mathsf{then}\ \mathsf{V}\ (u\ \#\ l\ !\ 1)\ \mathsf{else}\ \mathsf{if}\ 1 = \mathsf{Suc}\ n\ \mathsf{then}\ f\, l\ \mathsf{else}\ \mathsf{V}\ (u\ \#\ l\ !\ 0)) \\
\quad \mathsf{A} & (\mathsf{if}\ 5 < \mathsf{Suc}\ n\ \mathsf{then}\ \mathsf{V}\ (u\ \#\ l\ !\ 5)\ \mathsf{else}\ \mathsf{if}\ 5 = \mathsf{Suc}\ n\ \mathsf{then}\ f\, l\ \mathsf{else}\ \mathsf{V}\ (u\ \#\ l\ !\ 4))
\end{array}$$

Hence $E = F\ 0\ (\mathsf{V}\ (l\ !\ 8)) = \lambda\, v.\ \lambda\, l.\ \mathsf{L}\ u.\ (\mathsf{V}\ u)\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 8))\ \mathsf{A}\ (\mathsf{V}\ (l\ !\ 3))$ This will seem like a lot of work. Remember that for us, Isabelle HOL takes the strain! For the language given in this section, we can prove

Theorem 3 (Isabelle HOL). *The simple object level language is deterministic and enjoys subject reduction.*

A specific goal of our work is to investigate the viability of encoding and reasoning about effect based compiler transformations. We have chosen to study the MIL-lite language of Benton and Kennedy [2], although the application of our tools to MIL-lite is not, in itself, a central topic of this paper and it will be described it in detail in a forthcoming paper. The purpose of this section is to demonstrate the applicability of `Bsyntax`. MLj [3] is a SML-to-Java bytecode compiler, constructed through a typed intermediate language with effect-specific computation types, called

MIL. Benton and Kennedy identified a fragment of MIL, called MIL-lite, and have shown that it can be used to validate effect based transformations. MIL-lite is a non-trivial language, whose type system contains integers, integer references, functions, products, sums, and *effect based computations*. Moreover, a subtyping relation is induced by effect inclusion. The term system includes the expected machinery. We have encoded in Isabelle HOL the MIL-lite type system, and its evaluation relation, and proved that a subject reduction theorem holds.

5 A Presheaf Topos Model

We give a presheaf model of `Bsyntax`. As well as being interesting in its own right, a key point is that we show that it validates recursion over all types $var^n \Rightarrow eic$ by exhibiting such types "as" initial algebras—see Section 5.5. Of course, our definitional approach in Isabelle HOL should be (internally) consistent! This work provides additional justification for what we are doing. We work with a topos of presheaves $\mathcal{F}_w \stackrel{\text{def}}{=} \mathcal{S}et^{\mathbb{F}_\omega}$. In this section we show that there is an initial algebra $(exp, [\mathsf{V}, \mathsf{L}, \mathsf{A}])$ for the functor $T_{wh} : \mathcal{F}_w \to \mathcal{F}_w$, that is

$$T_{wh}\ exp = var + (var \Rightarrow exp) + exp^2 \xrightarrow{[\mathsf{V},\mathsf{L},\mathsf{A}]} exp$$

where of course *var* and *exp* are now objects (functors) of $\mathcal{F}_w$. Let us write $T_{sr}\ \xi \stackrel{\text{def}}{=} K_\omega + (var \Rightarrow \xi) + \xi^2$ and also $T_{sr'}\ \xi \stackrel{\text{def}}{=} K_\omega + \xi + \xi^2$, where K_ω is a constant functor in $\mathcal{F}_w$. We shall also show that the functor $eic \stackrel{\text{def}}{=} var^\omega \Rightarrow exp$ gives rise to initial algebras $T_{sr}\ eic \to eic$ and $T_{sr}(var^n \Rightarrow eic) \to var^n \Rightarrow eic$ and $T_{sr'}\ eic \to eic$ and $T_{sr'}(var^n \Rightarrow eic) \to var^n \Rightarrow eic$. This semantically validates the higher order substitution functions which are defined by primitive recursion over such types.

In the remainder of this section we proceed as follows. First, we give a collection of technical definitions and results[2] which underpins the results mentioned above. Then we prove the existence of an initial algebra in $\mathcal{F}_w$ for T_{wh}. We can then define the data $(exp, [\mathsf{V}, \mathsf{L}, \mathsf{A}])$ and show that $exp \cong \Omega$. Finally we show that *eic* and $var^n \Rightarrow eic$ are also initial algebras for both T_{sr} and $T_{sr'}$.

5.1 Some Supporting Definitions and Results

$\mathbb{F}_\omega$ is the full subcategory of $\mathcal{S}et$ whose objects are the Peano sets $0, 1, 2, \ldots$ and ω. We will use χ and ζ to range over arbitrary objects.

We will write $\mathcal{Y} : \mathcal{C}^{op} \to \mathcal{S}et^{\mathcal{C}}$ for the Yoneda embedding, with $\mathcal{Y}\ \xi \stackrel{\text{def}}{=} \mathcal{C}(\xi, -)$, where of course $F\xi \cong \mathcal{S}et^{\mathcal{C}}(\mathcal{Y}\ \xi, F)$ is the Yoneda isomorphism.

We will use the **shift functor**, $\delta : \mathcal{F}_w \to \mathcal{F}_w$, defined by $\delta\ \xi \stackrel{\text{def}}{=} \xi \circ (1 + (-))$. We often, as in this definition, identify objects A of categories with the corresponding identities id_A. The definition here is a minor adaptation of the shift functor of [10]. This functor is used to model the contractions of contexts by one variable which takes place when abstraction terms are formed in `Bsyntax`.

The presheaf *var* is defined by $var(\xi) \stackrel{\text{def}}{=} \xi$ where ξ is either an object or morphism of $\mathbb{F}_\omega$. Notice that *var* is also defined up to isomorphism by $\mathcal{Y}\ 1$ where $\mathcal{Y} : \mathbb{F}_\omega^{op} \to \mathcal{F}_w$, that is, by the embedding of the finite generic context consisting of a single variable.

We also need a number of lemmas and propositions. For space reasons these are all in the appendix.

[2] Most appear in the appendix, along with sketch proofs.

5.2 Obtaining an Initial Algebra for T_{wh} in $\mathcal{F}_w$

Lemma 5 tells us that $var \Rightarrow G \cong \delta\ G$ for any G. Hence we can find an initial algebra for T_{wh} by instead finding an initial algebra (Ω, σ) for the functor $\xi \mapsto var + (\delta\ \xi) + \xi^2$. We could show that Ω exists using an adaptation of the traditional methods expressing Ω as a colimit of a certain chain. Here, we proceed directly. We define $S_0 \stackrel{\text{def}}{=} \varnothing$, the empty presheaf, and set $S_{r+1} \stackrel{\text{def}}{=} var + (\delta\ S_r) + S_r^2$, giving a family of presheaves $(S_r \mid r \geq 0)$. It is easy to check that there are subobjects $i_r : S_r \hookrightarrow S_{r+1}$, so that we can define $\Omega \stackrel{\text{def}}{=} \bigcup_r S_r$ by Lemma 6. Some of the remaining details are contained in the appendix, page 17, and related examples can be found in [5].

5.3 Defining the Algebra $(exp, [\mathsf{V}, \mathsf{L}, \mathsf{A}])$

Now we can define the presheaf *exp* in $\mathcal{F}_w$. We will deal with Isabelle HOL variables of types $e :: var^n \Rightarrow eic$, $l :: var\ stream$, $f :: nat \Rightarrow nat$. We will also regard morphisms $\rho^\dagger : \omega \to \omega$ in $\mathbb{F}_\omega$ as Isabelle HOL variables of type $nat \Rightarrow nat$ (see Lemma 2). We let k range over the natural numbers. The function occ is defined *by primitive recursion* in `Bsyntax`, and $\mathsf{occ}\ k\ (\lambda\ l.\ e\ l)$ indicates that $\mathsf{V}\ (l\ !\ k)$ occurs in $\lambda\ l.\ e\ l$. On an object χ we define

$$exp\ \chi \stackrel{\text{def}}{=} \{\lambda\ l.\ e\ (l \circ f) \mid \mathsf{prop}\ (\lambda\ l.\ e\ l)\ \wedge\ \forall k \geq \chi(\neg \mathsf{occ}\ k\ (\lambda\ l.\ e\ (l \circ f)))\}$$

which is well defined by Lemma 4. The idea, roughly speaking, is that $exp\ \chi$ is the set of proper terms whose variables are all projections which are strictly less than χ. On a morphism $\rho : \chi \to \zeta$ we set

$$(exp\ \rho)(\lambda\ l.\ e\ (l \circ f)) \stackrel{\text{def}}{=} \lambda\ l.\ e\ (l \circ \rho^\dagger \circ f)$$

Note that this does indeed define a functor! This follows from the restriction $\forall k \geq \chi(\neg \mathsf{occ}\ k\ (\lambda\ l.\ e\ (l \circ f))$, together with the simple fact that if $k < \chi$ and $\rho_1 : \chi \to \zeta$ and $\rho_2 : \zeta \to \zeta'$, then by Lemma 2 we have $(\rho_2 \circ \rho_1)^\dagger(k) = \rho_2 \circ \rho_1(k) = {\rho_2}^\dagger \circ {\rho_1}^\dagger(k)$.

We define the natural transformations alluded to on page 9. $\mathsf{V} : var \to exp$ has components $\mathsf{V}_\chi : var\ \chi \to exp\ \chi$ given by $\mathsf{V}(i) \stackrel{\text{def}}{=} \lambda\ l.\ \mathsf{V}\ (l\ !\ i)$ where $i < \chi$. Next, $\mathsf{L} : \delta\ exp \to exp$ has components $\mathsf{L}_\chi : exp(1 + \chi) \to exp\ \chi$ given by

$$\mathsf{L}_\chi(\lambda\ l.\ e\ (l \circ f)) \stackrel{\text{def}}{=} \lambda\ l.\ \mathsf{L}\ u.\ \widehat{e}\ u\ (l \circ (\lambda\ k.\ f(k) - 1))$$

where we make use of Lemma 3 to define $\widehat{e}$. Finally, natural transformation $\mathsf{A} : exp^2 \to exp$ has components $\mathsf{A}_\chi : (exp\ \chi)^2 \to exp\ \chi$ given by

$$\mathsf{A}_\chi(\lambda\ l.\ e_1\ (l \circ f), \lambda\ l.\ e_2\ (l \circ f)) \stackrel{\text{def}}{=} \lambda\ l.\ (e_1\ (l \circ f))\ \mathsf{A}\ (e_2\ (l \circ f))$$

See the appendix, page 17, for the interesting case of naturality of L which depends crucially on our choice of coproducts in $\mathbb{F}_\omega$.

5.4 Proving Initiality of $(exp, [\mathsf{V}, \mathsf{L}, \mathsf{A}])$

We now show that the presheaf algebra Ω is isomorphic to the presheaf *exp*. We show that there are natural transformations $\phi : \Omega \to exp$ and $\psi : exp \to \Omega$, such that for any χ in $\mathbb{F}_\omega$, the functions ϕ_χ and ψ_χ give rise to a bijection between $\Omega\ \chi$ and $exp\ \chi$. To specify $\phi : \Omega \to exp$ we define a family of natural transformations $\phi_r : S_r \to exp$, and appeal to Lemma 7.

– $\phi_0 : S_0 = \varnothing \to exp$ has as components the empty function, and
– recursively we define

$$\phi_{r+1} \stackrel{\text{def}}{=} [\mathsf{V}, \mathsf{L} \circ \delta\ \phi_r, \mathsf{A} \circ \phi_r^2] : S_{r+1} = var + \delta\ S_r + S_r^2 \to exp$$

To specify $\psi : exp \to \Omega$, for any χ in $\mathbb{F}_\omega$ we define functions $\psi_\chi : exp\ \chi \to \Omega\ \chi$ as follows. First note that $S_r\chi \subset \Omega\chi$ for any r by definition of Ω. Then we define

– $\psi_\chi(\lambda\, l.\, \mathsf{V}\ ((l \circ f)\ !\ i)) \stackrel{\text{def}}{=} (f(i), 1) \in S_1\chi = \chi \times \{1\}$.
– $\psi_\chi(\lambda\, l.\, \mathsf{L}\ u.\ e\ u\ (l \circ f)) \stackrel{\text{def}}{=} \mathsf{in}_{S_r(1+\chi)}(\psi_{1+\chi}(\lambda\, l.\, e\ ((l\ !\ 0))\ (\mathsf{tl}\ (l \circ f))))$ where $r \geq 0$ is the rank of $\lambda\, l.\, \mathsf{L}\ u.\ e\ u\ (l \circ f)$.
– $\psi_\chi(\lambda\, l.\, e_1\ (l \circ f)\ \mathsf{A}\ e_2\ (l \circ f)) \stackrel{\text{def}}{=} \mathsf{in}_{(S_r\chi)^2}((\psi_\chi(\lambda\, l.\, e_1\ (l \circ f)), \psi_\chi(\lambda\, l.\, e_2\ (l \circ f))))$ where $r \geq 0$ is the rank of $\lambda\, l.\, e_1\ (l \circ f)\ \mathsf{A}\ e_2\ (l \circ f)$.

Some of the remaining details are in the appendix, page 18.

5.5 Validating Higher Order Recursion Principles

Summarizing, we have an initial algebra

$$T_{wh}(exp) = var + (var \Rightarrow exp) + exp^2 \xrightarrow{[\mathsf{V}, \mathsf{L}, \mathsf{A}]} exp$$

in the category $\mathcal{F}_w$. We define the presheaf $eic \stackrel{\text{def}}{=} var^\omega \Rightarrow exp$. Note that by Proposition 2, the presheaf $var^\omega \Rightarrow (-)$ preserves all colimits. It obviously has a left adjoint, so preserves all limits. Hence we have the following

$$\begin{aligned} eic &\cong var^\omega \Rightarrow (var + (var \Rightarrow exp) + exp^2) \\ &\cong (var^\omega \Rightarrow var) + (var^\omega \Rightarrow (var \Rightarrow exp)) + var^\omega \Rightarrow exp^2 \\ &\cong (var^\omega \Rightarrow var) + var \Rightarrow (var^\omega \Rightarrow exp) + (var^\omega \Rightarrow exp)^2 \\ &\cong K_\omega + (var \Rightarrow eic) + eic^2 \\ &\cong K_\omega + eic + eic^2 \end{aligned}$$

where the penultimate isomorphism follows by calculating with Lemma 5. Moreover, if in Proposition 1 we take $T' \stackrel{\text{def}}{=} T_{sr}$, $L \stackrel{\text{def}}{=} var^\omega \Rightarrow (-)$ and $T \stackrel{\text{def}}{=} T_{wh}$, we see that eic is an initial algebra for T_{sr}. Similarly, we can see that eic is also an initial algebra for $T_{sr'}$. Finally, note that $var^n \Rightarrow (-)$ also has both left and right adjoints, so $var^n \Rightarrow eic$ is also an initial algebra for the same functors.

6 Related Work

The idea of viewing *open* terms as functions on arbitrary numbers of variables implemented via streams originally appeared in [7]. The only reported experiment was the adequacy of the translation between the second order weak HOAS syntax and a first-order one, with the induction principle being provided by the `valid` predicate. Later, possibly due to the lack of extensional equality in Coq, [6] abandoned this track to revert to standard weak HOAS. The term-in-context style of encoding was resurrected in McDowell's thesis [14] to handle proofs by induction over open terms in a two-level approach, such as type uniqueness [15]. Miller noted the same problems

with respect to the encoding of properties of the π-calculus such as bisimulation [19] which eventually led him to internalize this behavior with a new universal quantifier ∇ operating over *local* signatures [20].

Miller was also the first one to investigate functional programming over HOAS [18]. Here the idea is to enrich a language such as SML with the capability to directly handle data involving variable binding, abstraction and higher-order pattern matching on bound variables. One related outcome is the *FreshML* language by Pitts and Gabbay [28], which is a full-fledged functional language which additionally provides a very elegant and semantically sound [11] way to program modulo α-conversion. Programs are checked for *freshness* of object bound names and are promoted only if used in ways that are insensitive to renaming. Many other features are present (see `www.freshml.org`); we just remark that abstractions act over pairs "atom, expression" and capture-avoiding substitution is easily programmed over a user-defined data type with variable binding. Gabbay is exploring same ideas in the context of a logical framework [12].

Schürmann [26] proposes functional programming with full HOAS via a two-level approach: the Edinburgh Logical Framework provides the data representation language, and a meta type-theory $\mathcal{M}_\omega^+$ supports programming with pattern matching and recursion. The crucial problem of recursion over open LF terms is solved by the notion of *regular world* which captures the predictable way a datatype with variable bindings is extended when descending by recursion into the binding cases. This idea is also at the heart of *Twelf*, the meta-logical framework which allows induction over full HOAS [25].

A one-level approach based on a modal λ-calculus was instead suggested in [27] and somewhat refined in [8]. The aim is to provide a uniform system where one can define functions by case analysis and primitive recursion, while preserving the adequacy of full HOAS encodings. This is achieved by separating the parametric function space from the primitive recursive one with the *S*4 box operator, which classifies those *closed* terms over which one can iterate or distinguish cases. This approach, albeit elegant, has not been implemented yet.

These approaches differ from ours in that they involve re-engineering of the logical framework. Our work has the advantage that it remains within (classical) Isabelle HOL.

7 Conclusions

We solved the problem of how to define a combinator for primitive recursion over HOAS by adapting some known techniques (described in the previous Section) to produce a type *eic* of λ-calculus terms-in-infinite-contexts. This type satisfies the isomorphism $var \Rightarrow eic \cong eic$ which crucially enables recursion under binders by creating and internalizing a closed world assumption for "traditional" terms-in-context (open terms). The combinator allows us to define functions directly—in a theorem prover one often has to show that relations are total and functional. Here, the proof of existence of the combinator subsumes such labour. We have developed an interesting topos model, and showed that we can exhibit the types over which we perform recursion as initial algebras in the topos model. We have applied our work to the encoding of, and reasoning about, a substantial object level language.

In the proof of subject reduction for MIL-lite there are various lemmas for the weakening and substitution properties of the typing judgment. These are proved via conventional arguments about the injective relabelling of free variables. These proofs are elegantly expressed in `Bsyntax` since relabelling amounts to pre-composition of a stream $l :: nat \Rightarrow var$ with a function $r :: nat \Rightarrow nat$. In order to determine how well our techniques scale it would be interesting to see if these proofs could be made generic or proved 'once and for all' for suitable equivariant predicates [11].

To reason about object logics we will need to establish principles of induction and primitive recursion over syntax defined by an arbitrary binding signature. Given the work presented here, the derivation of such principles should be routine and we hope to implement them as an Isabelle HOL package similar to the current datatypes package. Moreover, we shall investigate the possibility of defining functions by *cases* over HOAS and by *well-founded* recursion.

The combinator for primitive recursion has been developed in a framework of weak HOAS (where variables are of type *var* rather than of type *exp*). In principle, there is no reason why the same approach could not be applied in the full HOAS framework of `Hybrid` [1], together with an extension of the categorical models. There are some technical details to be sorted out but it should be possible to prove in `Hybrid` that substitution defined by primitive recursion coincides (for proper terms) with the meta-level β-equality of Isabelle. The full HOAS notion of terms-in-contexts can be used to implement Miller & Tiu's ∇ logic [20], where local signatures are seen as contexts, e.g. $\sigma \triangleright B$ as $\lambda\sigma.\, B$.

Finally, nothing prevents us from implementing the static and dynamic semantics of MIL-lite according to the two-level approach [9, 23], that is encoding object-level environments, such as typing contexts, with hypothetical judgments in the meta-logic. Since MIL-lite has some imperative features, it would benefit of an encoding based on a linear specification logic, in the spirit of [15]. Further, the terms-in-context approach directly supports reasoning by induction over open terms; this is crucial when establishing the notion of program equivalence [24] which is used for the compiler optimizations in MIL-lite [2],

References

1. 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.
2. N. Benton and A. Kennedy. Monads, effects and transformations. *Electronic Notes in Theoretical Computer Science*, 26, 1999.
3. Nick Benton, Andrew Kennedy, and George Russell. Compiling Standard ML to Java bytecodes. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, volume 34(1), pages 129–140, 1999.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. R. L. Crole. Basic Category Theory for Models of Syntax. Proceedings of the Summer School on Generic Programming, Oxford, UK, August 2002. 40 pp with index. To appear in LNCS, planned 2003.
6. Joëlle Despeyroux, Amy Felty, and André 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, April 1995. Springer-Verlag LNCS 902.
7. Joëlle Despeyroux and André Hirschowitz. Higher-order abstract syntax with induction in Coq. In Frank Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, pages 159–173, Kiev, Ukraine, July 1994. Springer-Verlag LNAI 822.
8. Joëlle Despeyroux and Pierre Leleu. Recursion over objects of functional type. *Mathematical Structures in Computer Science*, 11(4):555–572, 2001.
9. Amy Felty. Two-level meta-reasoning in Coq. 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.
10. Marcelo Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proc. of 14th Ann. IEEE Symp. on Logic in Computer Science, LICS'99, Trento, Italy, 2–5 July 1999*, pages 193–202. IEEE Computer Society Press, Los Alamitos, CA, 1999.

11. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.
12. Murdoch J. Gabbay. FM-HOL, a higher-order theory of names. In F. Kamareddine, editor, *35 Years of Automath*, http://www.cee.hw.ac.uk/~fairouz/automath2002/informal-proceedings, April 2002. Heriot-Watt University, Edinburgh, Scotland.
13. Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. of 14th Ann. IEEE Symp. on Logic in Computer Science, LICS'99, Trento, Italy, 2–5 July 1999*, pages 204–213. IEEE Computer Society Press, Los Alamitos, CA, 1999.
14. Raymond McDowell. *Reasoning in a Logic with Definitions and Induction*. PhD thesis, University of Pennsylvania, 1997.
15. Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.
16. James McKinna and Robert Pollack. Some lambda calculus and type theory formalized. *JAR*, 1998.
17. M. Miculan. Developing (meta)theory of lambda-calculus in the theory of contexts. In Simon Ambler, Roy Crole, and Alberto 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.
18. Dale Miller. An extension to ML to handle bound variables in data structures: Preliminary report. In *Informal Proceedings of the Logical Frameworks BRA Workshop*, June 1990.
19. Dale Miller. Encoding generic judgments: Preliminary results. In R.L. Crole S.J. Ambler and A. Momigliano, editors, *Electronic Notes in Theoretical Computer Science*, volume 58. Elsevier Science Publishers, 2001.
20. Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *Proceedings of the 18th Annual Symposium on Logic in Computer Science*, Ottawa, Canada, 2003.
21. E. Moggi. Notions of computation and monads. *Theoretical Computer Science*, 93:55–92, 1989.
22. A. Momigliano, S. J. Ambler, and R. L. Crole. A Comparison of Formalizations of the Meta-Theory of a Language with Variable Bindings in Isabelle. In Richard J. Boulton and Paul B. Jackson, editors, *Supplemental Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics*, pages 267–282, 2001. Report EDI-INF-RR-0046.
23. Alberto Momigliano and Simon Ambler. Multi-level meta-reasoning with higher order abstract syntax. In A. Gordon, editor, *FOSSACS*, volume 2620 of *LNCS*, pages 375–392. Springer Verlag, 2002.
24. Alberto Momigliano, Simon Ambler, and Roy Crole. A Hybrid encoding of Howe's method for establishing congruence of bisimilarity. *ENTCS*, 70(2), 2002.
25. Frank Pfenning and Carsten 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.
26. Carsten Schürmann. Recursion for higher-order encodings. In *Proceedings of Computer Science Logic (CSL 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 585–599, 2001.
27. Carsten Schürmann, Joëlle Despeyroux, and Frank Pfenning. Primitive recursion for higher-order abstract syntax. *Theoretical Computer Science*, 266(1–2):1–57, September 2001.
28. M. R. Shinwell and A. M. Pitts. *FreshML User Manual*. Cambridge University Computer Laboratory, November 2002. Available at ⟨`http://www.freshml.org/docs/`⟩.
29. M.B. Smyth and G.D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–783, 1982.

A Appendix: Some Lemmas and Propositions

We shall make crucial use of the fact that $\mathcal{F}_w$ has coproducts; in modeling `Bsyntax` we need a specified choice.

Lemma 1. *The category $\mathbb{F}_\omega$ has coproducts. The data in the proof constitute a specific choice.*

Proof. There are coproduct diagrams

$$n \xrightarrow{\mathsf{inl}} n+m \xleftarrow{\mathsf{inr}} m$$

where $\mathsf{inl}(i) \stackrel{\text{def}}{=} i$ and $\mathsf{inr}(j) \stackrel{\text{def}}{=} n+j$; and

$$n \xrightarrow{\mathsf{inl}} \omega \xleftarrow{\mathsf{inr}} \omega$$

where $\mathsf{inl}(i) \stackrel{\text{def}}{=} i$ and $\mathsf{inr}(j) \stackrel{\text{def}}{=} n+j$; and

$$\omega \xrightarrow{\mathsf{inl}} \omega \xleftarrow{\mathsf{inr}} \omega$$

where $\mathsf{inl}(i) \stackrel{\text{def}}{=} 2i$ and $\mathsf{inr}(j) \stackrel{\text{def}}{=} 2j+1$. Note the fact that in both cases, $\mathsf{inl}(0) = 0$. This will play a crucial role in modeling variable binding—recall the discussion at the end of Section 2 where it is pointed out that abstractions are formed over 0th variable projections. We will sometimes write $n+\omega$ or $\omega+\omega$ instead of ω if this makes the use of a canonical coproduct more transparent.

Each morphism ρ in $\mathbb{F}_\omega$ will give rise to a re-labelling of variable projections, sending $\mathsf{V}\ (l\ !\ i)$ to $\mathsf{V}\ (l\ !\ \rho(i))$. In order to do this we will need to extend each ρ so that it may be composed with a stream l. The lemma below shows how to do this.

Lemma 2. *Suppose that $\rho : \chi \to \zeta$ is any $\mathbb{F}_\omega$ morphism. Then there is a canonical morphism $\rho^\dagger \stackrel{\text{def}}{=} [\iota \circ \rho, id_\omega] : \omega \to \omega$ which extends the source of ρ if χ is finite, that is for all $i < \chi$, $\rho^\dagger(i) = \rho(i)$ and otherwise $\rho^\dagger(i) = i$, where $\iota : \zeta \to \omega$.*

The next lemma will be used in the definition of the categorical analogue of `Bsyntax` abstraction, allowing contexts to be expanded or contracted

Lemma 3. *For any m and n we have $var^n \Rightarrow eic \cong var^m \Rightarrow eic$ given by a mapping $\xi \mapsto \widehat{\xi}$ which is defined in the proof.*

Proof. Without loss of generality, suppose that $m = n+\varepsilon$ with $\varepsilon \geq 1$ (if not swap n and m). Define isomorphism witnesses by setting

$$\lambda\, v_0\ \ldots\ v_{n-1}\ l.\ e\ v_0\ \ldots\ v_{n-1}\ l \mapsto \lambda\, v_0\ \ldots\ v_{n-1}\ u_0\ \ldots\ u_{\varepsilon-1}\ l.\ e\ v_0\ \ldots\ v_{n-1}\ (u_0\ \#\ \ldots\ \#\ u_{\varepsilon-1}\ \#\ l)$$

and

$$\lambda\, v_0\ \ldots\ v_{m-1}\ l.\ e\ v_0\ \ldots\ v_{m-1}\ l \mapsto \lambda\, v_0\ \ldots\ v_{n-1}\ l.\ e\ v_0\ \ldots\ v_{n-1}\ (l\ !\ 0)\ \#\ \ldots (l\ !\ \varepsilon-1)\ \#\ (\mathsf{dr}^\varepsilon\ l)$$

The easy proof involves simple reasoning up to $\beta\eta$ equality.

Lemma 4 ensures the re-labelling mentioned above to preserve properness of terms.

Lemma 4. *If $f :: var \Rightarrow var$ and* prop $(\lambda\, l.\ e\ l)$*, then* prop $(\lambda\, l.\ e\ (l \circ f))$.

Lemma 5 is used in the proof of Proposition 2, and at the end of Section 5.

Lemma 5. *Let G be a presheaf. Then for any χ in $\mathbb{F}_\omega$ we have a natural isomorphism $var^\chi \Rightarrow G \cong \delta^\chi\ G$ where $var^\chi \stackrel{\text{def}}{=} \Pi_{i \in \chi} var$ is a product in the presheaf category $\mathcal{F}_w$.*

Proof. We have

$$(var^{\chi} \Rightarrow G)(\zeta) \cong \mathcal{F}_w((\mathcal{Y}\,\zeta) \times (\mathcal{Y}\,\chi), G) \cong \mathcal{F}_w(\mathcal{Y}\,(\zeta+\chi), G) \cong G(\chi+\zeta)$$

where the first isomorphism follows from the definition of exponentials and the simple fact that $var^{\chi} \cong \mathcal{Y}\,\chi$ in $\mathcal{F}_w$, the second from the universal property of coproducts, and the final isomorphism is Yoneda together with a simple isomorphism of coproducts.

Proposition 1 is used in the proof of the existence of various initial algebras.

Proposition 1. *Let $T, T', L, R \colon \mathcal{C} \to \mathcal{C}$ be functors, such that $L \dashv R$, and $\phi \colon T' \circ L \cong L \circ T$ naturally. If (Ω, σ) is an initial object in the category $\mathcal{C}^T$ of T-algebras, then $(L\Omega, L\sigma \circ \phi_\Omega)$ is initial in $\mathcal{C}^{T'}$.*

Proof. We can define functors $L^T : \mathcal{C}^T \to \mathcal{C}^{T'}$ and $R^{T'} : \mathcal{C}^{T'} \to \mathcal{C}^T$ by setting

$$L^T(A, \sigma_A) \stackrel{\text{def}}{=} (LA, L\sigma_A \circ \phi_A)$$
$$R^{T'}(B, \sigma_B) \stackrel{\text{def}}{=} (RB, R\sigma_B \circ \overline{T'(\varepsilon_{RB}) \circ \phi^{-1}_{RB}})$$

on objects, with the expected extension to morphisms. It is a tedious exercise to show that $L^T \dashv R^{T'}$. Hence L^T preserves all colimits and hence $L^T(\Omega, \sigma)$ is an initial T'-algebra as required.

Proposition 2 shows that the functor $var^n \Rightarrow (-)$ is itself a left adjoint, and hence it can be used as an instance of L in Proposition 1.

Proposition 2. *The functor $var^{\chi} \Rightarrow (-) : \mathcal{F}_w \to \mathcal{F}_w$ has a right adjoint R given on objects and morphisms by $RF(\xi) \stackrel{\text{def}}{=} \mathcal{F}_w(var^{\chi} \Rightarrow \mathcal{Y}\,\xi, F)$.*

Proof. We have to give a natural bijection $\mathcal{F}_w(var^{\chi} \Rightarrow G, F) \cong \mathcal{F}_w(G, RF)$. Note that from Lemma 5 we have $var^{\chi} \Rightarrow \mathcal{Y}\,\zeta \cong \mathbb{F}_\omega(\zeta, \chi + (-))$. If $\alpha \colon G \to RF$ in $\mathcal{F}_w$, then we have

$$(\alpha_\zeta : G\zeta \to \mathcal{F}_w(\mathbb{F}_\omega(\zeta, \chi + (-)), F) \mid \zeta \in \mathbb{F}_\omega)$$

and we define the mate across the adjunction by

$$(\tilde{\alpha}_\zeta : G(\chi + \zeta) \to F\zeta \mid \zeta \in \mathbb{F}_\omega)$$

by

$$(\tilde{\alpha}_\zeta(x) \stackrel{\text{def}}{=} \alpha_{\chi+\zeta}(x)_\zeta(id_{\chi+\zeta}) \mid \zeta \in \mathbb{F}_\omega)$$

The remaining details are omitted.

The final two lemmas of this section are minor modifications of standard results used in the construction of initial algebras as (co)limits of diagrams of chains [29].

Lemma 6. *Suppose that $(S_r \mid r \geq 0)$ is a family of presheaves in $\mathcal{F}_w$, with $i_r : S_r \hookrightarrow S_{r+1}$ for each r. Then there is a **union** presheaf U in $\mathcal{F}_w$, such that $i'_r : S_r \hookrightarrow U$. We sometimes write $\cup_r S_r$ for U.*

Proof. On objects χ of $\mathbb{F}_\omega$ we define $U\chi \stackrel{\text{def}}{=} \bigcup_r S_r\chi$. On morphisms $\rho : \chi \to \zeta$ in $\mathbb{F}$ we define the function $U\rho : U\chi \to U\zeta$ by setting $(U\rho)(\xi) \stackrel{\text{def}}{=} (S_r\rho)(\xi)$ where $\xi \in U\chi$, and r is any index for which $\xi \in S_r(\chi)$.

Lemma 7. *Let $(\phi_r : S_r \to A \mid r \geq 0)$ be a family of natural transformations in $\mathcal{F}_w$ with the S_r as in Lemma 6, and such that $\phi_{r+1} \circ i_r = \phi_r$. Then there is a unique natural transformation $\phi : U \to A$, such that $\phi \circ i'_r = \phi_r$.*

Proof. The proof requires a simple calculation using the definitions. Note that there are functions $\phi_\chi : U\chi \to A\chi$ where we set $\phi_\chi(\xi) \stackrel{\text{def}}{=} (\phi_r)_\chi(\xi)$ for $\xi \in S_r\chi$. The conditions of the lemma (trivially) imply the existence and uniqueness of the ϕ_χ, which are natural in χ.

Obtaining an Initial Algebra for T_{wh} in $\mathcal{F}_w$

Next we consider the structure map $\sigma : var + \delta\,\Omega + \Omega^2 \to \Omega$. This natural transformation in $\mathcal{F}_w$ must be given by a cotupling of (insertion) natural transformations $\sigma \stackrel{\text{def}}{=} [\kappa, \kappa', \kappa'']$. For the first morphism, note that $var \cong S_1$, and we set $\kappa \stackrel{\text{def}}{=} i'_r \circ \cong$ where $i'_r : S_r \hookrightarrow U$. We define κ'' by applying Lemma 7 to the family of morphisms

$$\kappa''_r : S_r \xrightarrow{\quad \mathsf{in}_{S_r} \quad} var + S_r + S_r^2 = S_{r+1} \hookrightarrow \Omega$$

Finally, to define κ', note that $(\delta\, T)\xi \stackrel{\text{def}}{=} T(1+\xi) = \bigcup_r S_r(1+\xi) = \bigcup_r (\delta\, S_r)\xi = (\bigcup_r \delta\, S_r)\xi$. Hence we can apply an instance of Lemma 7 to the family of morphisms

$$\kappa'_r : \delta\, S_r \xrightarrow{\quad \mathsf{in}_{\delta\, S_r} \quad} var + \delta\, S_r + S_r^2 = S_{r+1} \hookrightarrow \Omega$$

Note that we must check that $\delta\, S_r \hookrightarrow \delta\, S_{r+1}$ for all r, by induction. The routine details are omitted.

We must verify that $\sigma : T_{wh}\Omega \to \Omega$ is an initial algebra. Consider

$$\begin{array}{ccc} var + \delta\,\Omega + \Omega^2 & \xrightarrow{\ \sigma\ } & \Omega \\ \Big\downarrow{\scriptstyle var + \delta\,\overline{\alpha} + \overline{\alpha}^2} & (*) & \Big\downarrow{\scriptstyle \overline{\alpha}} \\ var + \delta\, A + A^2 & \xrightarrow[\ \alpha\ ]{} & A \end{array}$$

To define $\overline{\alpha} : \Omega \to A$ we specify a family of natural transformations $\overline{\alpha}_r : S_r \to A$ and appeal to Lemma 7. We define $\overline{\alpha}_0$ to be the natural transformation with components the empty functions $\varnothing : \varnothing \to A\chi$ for each χ in $\mathbb{F}_\omega$, and

$$\overline{\alpha}_{r+1} \stackrel{\text{def}}{=} [\alpha \circ \mathsf{in}_{var}, \alpha \circ \mathsf{in}_A \circ \delta\,\overline{\alpha}_r, \alpha \circ \mathsf{in}_{A^2} \circ \overline{\alpha}_r^2] : S_{r+1} = var + S_r + S_r^2 \to A$$

The verification that $(*)$ commutes is omitted.

Naturality of L

Naturality is the requirement that for any $\rho : \chi \to \zeta$ in $\mathbb{F}_\omega$, the diagram below commutes

$$\begin{array}{ccc} (\delta\, exp)\chi = exp(1+\chi) & \xrightarrow{\ \mathsf{L}_\chi\ } & exp\,\chi \\ \Big\downarrow{\scriptstyle (\delta\, exp)\rho = exp(1+\rho)} & & \Big\downarrow{\scriptstyle exp\,\rho} \\ (\delta\, exp)\zeta = exp(1+\zeta) & \xrightarrow[\ \mathsf{L}_\zeta\ ]{} & exp\,\zeta \end{array}$$

It does commute, as seen from the following calculation

$$\begin{array}{ccc} \lambda\, l.\, e\,(l \circ f) & \xrightarrow{\ \mathsf{L}_\chi\ } & \lambda\, l.\, \mathsf{L}\, u.\, \widehat{e}\, u\,(l \circ (\lambda\, k.\ f(k)-1)) \\ \Big\downarrow{\scriptstyle exp(1+\rho)} & & \Big\downarrow{\scriptstyle exp\,\rho} \\ \lambda\, l.\, e\,(l \circ (1+\rho)^\dagger \circ f) & \xrightarrow[\ \mathsf{L}_\zeta\ ]{} & \lambda\, l.\, \mathsf{L}\, u.\, \widehat{e}\, u\,(l \circ (\lambda\, k.\ \rho(f(k)-1))) \end{array}$$

whose proof requires a straightforward calculation, and Lemma 1 which specifies coproducts in $\mathbb{F}_\omega$. The key point here is that in forming the abstractions via L_ζ, any variable projection index k for which $f(k) = 0$ will be abstracted, as $(1+\rho)^\dagger(f(k)) = (1+\rho)(f(k)) = 0$. Otherwise $f(k) = 1+j$ for some j, and then $\lambda\, k.\ ((1+\rho)^\dagger \circ f)(k) - 1 = \lambda\, k.\ (\rho(f(k)-1)+1) - 1$ using Lemma 1 and Lemma 2.

Proving Initiality of $(exp, [\mathsf{V}, \mathsf{L}, \mathsf{A}])$

We should verify that for any χ in $\mathbb{F}_\omega$, there is a natural bijection

$$\Omega\,\chi \underset{\psi_\chi}{\overset{\phi_\chi}{\underset{\cong}{\rightleftarrows}}} exp\,\chi$$

We shall just show that $\psi_\chi \circ \phi_\chi = id_{\Omega\,\chi}$ and omit the other details. Suppose that $\xi \in S_r\chi \subset \Omega\chi$. Then by definition, $\psi_\chi(\phi_\chi(\xi)) = \psi_\chi((\phi_r)_\chi(\xi))$. We show by induction that for all $r \geq 0$, if ξ is any element in level r and χ any object of $\mathbb{F}_\omega$, then $\psi_\chi((\phi_r)_\chi(\xi)) = \xi$. For $r = 0$ the assertion is vacuously true, as $S_0\chi$ is always empty. We assume the result holds for any $r \geq 0$. Let $\xi \in S_{r+1}\chi = var\,\chi + S_r(1+\chi) + S_r\chi^2$. Then we have

$$\psi_\chi((\phi_{r+1})_\chi(\xi)) = \psi_\chi([\mathsf{V}_\chi, \mathsf{L}_\chi \circ (\phi_r)_{1+\chi}, \mathsf{A}_\chi \circ (\phi_r)^2_\chi](\xi))$$

We can complete the proof by analyzing the cases which arise depending on which component ξ lives in. We just consider the case when $\xi = \mathsf{in}_{S_r(1+\chi)}(\xi')$ for some $\xi' \in S_r(1+\chi)$. We have

$$\begin{aligned}
\psi_\chi((\phi_{r+1})_\chi(\xi)) &= \psi_\chi((\mathsf{L}_\chi \circ (\phi_r)_{1+\chi})(\xi')) && (1)\\
&= \psi_\chi(\lambda\, l.\ \mathsf{L}\ u.\ \widehat{(\phi_r)_{1+\chi}}(\xi')\ u\ l) && (2)\\
&= \mathsf{in}_{S_r(1+\chi)\chi}(\psi_{1+\chi}(\lambda\, l.\ \widehat{(\phi_r)_{1+\chi}}(\xi')\ (l\ !\ 0)\ (\mathsf{tl}\ l))) && (3)\\
&= \mathsf{in}_{S_r(1+\chi)\chi}(\psi_{1+\chi}((\phi_r)_{1+\chi}(\xi'))) && (4)\\
&= \mathsf{in}_{S_r(n+1)\chi}(\xi') && (5)\\
&= \xi && (6)
\end{aligned}$$

where equation 5 follows by induction.