HIGHER–ORDER ALGEBRA WITH TRANSFINITE TYPES

L.J. Steggles
Department of Computer Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU
email: L.J.Steggles@ncl.ac.uk

Abstract

We extend the simple type system of higher–order algebra with transfinite types. We present a general model theory for transfinite higher–order algebra including results on the existence and construction of free and initial models, and a sound and complete equational calculus. We demonstrate the use of transfinite types for modelling polymorphism by specifying a simple polymorphic functional programming language.

1 Introduction

Higher–order algebra provides a natural framework in which to formally develop computing systems and has been shown to be substantially more expressive than first–order algebraic methods (see Kosiuczenko and Meinke [1994] and Meinke [1995]). For an introduction to higher–order algebraic methods see Möller [1987], Möller et al [1988] and Meinke [1992], while for examples of their applications see for example Meinke and Steggles [1994], Meinke [1994] and Steggles [1995].

However one of the limitations of higher–order algebra using only binary product and function types is the difficulty in modelling objects which are parametric in type, such as polymorphic functions, partial functions, generic data structures and infinite families of algorithms and architectures. We propose to overcome this limitation by extending the type system of higher–order algebra with *limit types*. The idea is that given a set of basic types we close up under product and function types until we reach the limit at which point we add a new type, a so called limit type. We then iterate this process to produce a transfinite type hierarchy in which the limit types act as "universal" types in which all the types below them in the transfinite type hierarchy can be embedded. We refer to the resulting theory as *higher–order algebra with transfinite types*. Following the approach of Meinke [1992], we develop a general model theory for higher–order algebra with transfinite types using the framework of many–sorted first–order predicate calculus (usually referred to as finite or simple type theory). Results presented include the existence and construction of free and initial models, and a sound and complete equational calculus. Higher–order algebra with transfinite types provides a natural framework in which to model polymorphism, partiality and error handling. We demonstrate this by considering the specification of a simple polymorphic functional programming language based on the language $\mathcal{PR}$ of primitive recursive functions presented in Thompson [1987]. This specification case study generalises the definitions and results of Meinke [1994].

The structure of this paper is as follows. In Section 2 we introduce a notation for transfinite types and extend the basic definitions of higher–order algebra to take account of transfinite types. In Section 3 we introduce the first–order condition of extensionality, present a representation theorem for extensional algebras and consider those algebraic constructions which preserve or induce extensionality. Section 4 considers the existence of free transfinite higher–order algebras and gives an existence theorem for free algebras in a class of transfinite higher–order algebras. We also consider the existence of initial extensional models and the conditions necessary for a class of extensional algebras to contain an initial extensional model. In Section 5 we present a sound and complete transfinite higher–order equational calculus and give a concrete construction of the initial extensional model for an equational theory. In Section 6 we demonstrate a possible application of transfinite types by specifying a simple polymorphic functional programming language. This specification case study demonstrates how transfinite types can be used to model partial functions and both parametric and overloading polymorphism. Finally, in Section 7 we make some concluding remarks about the results we have presented.

We have attempted to make this paper self contained. For further background material on universal algebra we recommend Cohn [1965], Wechler [1992] and Meinke and Tucker [1993].

2 Transfinite Higher–Order Algebras

In this section we present the syntax and semantics of higher–order algebras with transfinite types. We extend the $\times$ and $\rightarrow$ type system of higher–order algebra (see for example Meinke [1992]) with *limit types* and reformulate the definitions of a type structure S, an S–typed signature Σ and an S–typed Σ algebra. We refer to the resulting theory as *higher order algebra with transfinite types* (or simply *transfinite higher–order algebra*). A simple example of a transfinite higher–order algebra is presented at the end of the section. We begin by recalling some basic definitions of many–sorted first–order universal algebra (see for example Meinke and Tucker [1993]).

By a set S of sorts we mean any non-empty set. Let S^* denote the set of all words in the free monoid generated by S, let λ denote the empty word and define $S^+ = S^* - \{\lambda\}$. An S-sorted signature Σ is an $S^* \times S$-indexed family of sets $\Sigma = \langle\ \Sigma_{w,s} \mid w \in S^*,\ s \in S\ \rangle$. For $\lambda \in S^*$ and each sort $s \in S$, each element $c \in \Sigma_{\lambda,s}$ is a constant symbol of sort s. For each non-empty word $w = s(1)\dots s(n) \in S^+$ and each sort $s \in S$, each element $f \in \Sigma_{w,s}$ is a function symbol of domain type w, codomain type s and arity n.

Let Σ be an S-sorted signature. An S-sorted Σ algebra is an ordered pair (A, Σ^A), consisting of an S-indexed family $A = \langle\ A_s \mid s \in S\ \rangle$ of carrier sets A_s and an $S^* \times S$-indexed family $\Sigma^A = \langle\ \Sigma^A_{w,s} \mid w \in S^*,\ s \in S\ \rangle$ of sets of constants and algebraic operations. For each sort $s \in S$, $\Sigma^A_{\lambda,s} = \langle\ c_A \mid c \in \Sigma_{\lambda,s}\ \rangle$, where $c_A \in A_s$ is a constant that interprets c in A_s. For each $w = s(1)\dots s(n) \in S^+$ and $s \in S$, $\Sigma^A_{w,s} = \langle\ f_A \mid f \in \Sigma_{w,s}\ \rangle$, where $f_A : A^w \rightarrow A_s$ is an operation with domain $A^w = A_{s(1)} \times \cdots \times A_{s(n)}$ and codomain A_s which interprets f over A. As usual, we let A denote both a Σ algebra and its S-indexed family of carrier sets.

Let $A = \langle\ A_s \mid s \in S\ \rangle$ and $B = \langle\ B_s \mid s \in S\ \rangle$ be S-indexed families of sets then the basic set theoretic operations can be defined pointwise on A and B. We let $A \subseteq B$ denote pointwise inclusion, $A_s \subseteq B_s$, for each sort $s \in S$ and $A \subset B$ denotes the fact that $A \subseteq B$ but $A \neq B$. When no ambiguity arises we let $\emptyset$ denote the unique S-indexed family of empty sets. With A we associate an S-indexed family of cardinals, $|A| = \langle\ |A|_s = |A_s| \mid s \in S\ \rangle$ and $|A| \leq |B|$ denotes the fact that $|A|_s \leq |B|_s$ for each $s \in S$. We let $\phi : A \rightarrow B$ denote the S-indexed family of mappings $\phi = \langle\ \phi_s : A_s \rightarrow B_s \mid s \in S\ \rangle$. We say that ϕ is injective (respectively surjective, bijective) if, and only if, each ϕ_s is injective (respectively surjective, bijective). If $C = \langle\ C_s \mid s \in S\ \rangle$ is an S-indexed family of sets and $\phi : A \rightarrow B$ and $\psi : B \rightarrow C$ are S-indexed families of mappings then $\psi \circ \phi : A \rightarrow C$ denotes the pointwise composition

$$\psi \circ \phi = \langle\ (\psi \circ \phi)_s = \psi_s \circ \phi_s : A_s \rightarrow C_s \mid s \in S\ \rangle.$$

Next we recall some basic definitions for ordinals (see for example Devlin [1979] or Henle [1986]). Let *Ord* denote the (proper) class of von Neumann ordinals. Let $\emptyset \in Ord$ denote the *initial ordinal* and for any ordinal $\alpha \in Ord$ let $\alpha + 1$ denote the *successor ordinal* $\alpha \cup \{\alpha\} \in Ord$. All other ordinals $\beta \in Ord$ are called *limit ordinals*, the first being $\omega = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots\}$, the next $\omega + \omega$ and so on. Recall that *Ord* is well–ordered by $\subseteq$. Let $\alpha \in Ord$ be a limit ordinal, if $\alpha \neq \omega$ then define $\alpha^- \in Ord$ to be the greatest limit ordinal β such that $\beta < \alpha$, otherwise if $\alpha = \omega$ then define $\alpha^- = \emptyset$.

We define the rules of type formation for transfinite type theories as follows.

2.1 Definition. By a *type basis* $\mathcal{B}$ we mean a non-empty set. Define the *transfinite type hierarchy* $H(\mathcal{B})$ generated by a type basis $\mathcal{B}$ to be the class

$$H(\mathcal{B}) = \bigcup_{\alpha \in Ord} H_\alpha(\mathcal{B})$$

where the sets $H_\alpha(\mathcal{B})$ are defined by transfinite induction as follows. For $\emptyset \in Ord$ the initial ordinal, define $H_\emptyset(\mathcal{B}) = \mathcal{B}$. For $\alpha + 1 \in Ord$ a successor ordinal, define

$$H_{\alpha+1}(\mathcal{B}) = H_\alpha(\mathcal{B}) \cup \{ \ (\sigma \times \tau), (\sigma \to \tau) \mid \sigma, \tau \in H_\alpha(\mathcal{B}) \ \}.$$

For $\beta \in Ord$ a limit ordinal, define

$$H_\beta(\mathcal{B}) = \bigcup_{\alpha < \beta} H_\alpha(\mathcal{B}) \cup \{\beta\}.$$

Each limit ordinal $\beta \in H(\mathcal{B})$ is termed a *limit type*. □

We can assign an order $O(\tau) \in Ord$ to each type $\tau \in H(\mathcal{B})$ as follows.

2.2 Definition. Given any type basis $\mathcal{B}$ we define the *order* $O(\tau) \in Ord$ of each type $\tau \in H(\mathcal{B})$ by induction on the complexity of types. Let $\tau \in \mathcal{B}$ be a basic type, then define $O(\tau) = \emptyset$. Let $\sigma, \tau \in H(\mathcal{B})$, then define

$$O(\sigma \times \tau) = sup\{O(\sigma), O(\tau)\}, \quad O(\sigma \to \tau) = sup\{O(\sigma), O(\tau)\} + 1.$$

Let $\beta \in H(\mathcal{B})$ be a limit type, then define $O(\beta) = \beta$. □

In the sequel, we write $\sigma < \tau$ to denote that $O(\sigma) < O(\tau)$, for any types $\sigma, \tau \in H(\mathcal{B})$.

2.3 Definition. A (*transfinite*) *type structure* S over a type basis $\mathcal{B}$ is a subset of $H(\mathcal{B})$ which is closed under subtypes in the sense that: for any $\sigma, \tau \in H(\mathcal{B})$, if $(\sigma \times \tau) \in S$ or $(\sigma \to \tau) \in S$ then both $\sigma \in S$ and $\tau \in S$; and for each limit type $\beta \in S$ we have $\alpha \in S$, for each limit type $\alpha \in H(\mathcal{B})$ such that $\alpha < \beta$. □

For any ordinal $\alpha \in Ord$ and any type structure $S \subseteq H(\mathcal{B})$ over a type basis $\mathcal{B}$, S is said to be a *α–order type structure* if, and only if, the order of each type $\tau \in S$ is strictly less than α.

2.4 Definition. Let $S \subseteq H(\mathcal{B})$ be a type structure over a type basis $\mathcal{B}$. A (*transfinite*) *S-typed signature* Σ is an S-sorted signature such that: for each product type $(\sigma \times \tau) \in S$ we have two unary *projection operation symbols* $proj^{(\sigma \times \tau),1} \in \Sigma_{(\sigma \times \tau),\sigma}$ and $proj^{(\sigma \times \tau),2} \in \Sigma_{(\sigma \times \tau),\tau}$; for each function type $(\sigma \to \tau) \in S$ we have a binary *evaluation operation symbol* $eval^{(\sigma \to \tau)} \in \Sigma_{(\sigma \to \tau)\ \sigma,\tau}$;

and for each limit type $\beta \in S$ and for each type $\tau \in S$, such that $\beta^- \leq \tau < \beta$, we have a unary *injection operation symbol* $inj^\tau \in \Sigma_{\tau,\beta}$. $\square$

When the types σ and τ are clear we let $proj^1$ and $proj^2$ denote the projection operation symbols $proj^{(\sigma\times\tau),1}$ and $proj^{(\sigma\times\tau),2}$. An S-typed signature Σ is also termed an *α–order signature* when S is an α–order type structure. Next we introduce the intended interpretations of a transfinite higher–order signature.

2.5 Definition. Let $S \subseteq H(\mathcal{B})$ be a type structure over a type basis $\mathcal{B}$, let Σ be an S-typed signature and let A be an S-sorted Σ algebra. We say that A is *cumulative* if, and only if, for each product type $(\sigma \times \tau) \in S$ we have $A_{(\sigma\times\tau)} \subseteq A_\sigma \times A_\tau$, for each function type $(\sigma \to \tau) \in S$ we have $A_{(\sigma\to\tau)} \subseteq [A_\sigma \to A_\tau]$, and for each limit type $\beta \in S$ we have

$$\coprod_{\langle \tau\in S \ \mid \ \beta^- \leq \tau < \beta \rangle} A_\tau \ \subseteq A_\beta.$$

We say that A is a (*transfinite*) *S–typed Σ algebra* if, and only if, A is cumulative and
(i) for each product type $(\sigma \times \tau) \in S$ the operations

$$proj_A^{(\sigma\times\tau),1} : A_{(\sigma\times\tau)} \to A_\sigma, \quad proj_A^{(\sigma\times\tau),2} : A_{(\sigma\times\tau)} \to A_\tau,$$

are the *first* and *second projection operations*;
(ii) for each function type $(\sigma \to \tau) \in S$ the operation $eval_A^{(\sigma\to\tau)} : A_{(\sigma\to\tau)} \times A_\sigma \to A_\tau$ is the *evaluation operation*; and
(iii) for each limit type $\beta \in S$ and each type $\tau \in S$, such that $\beta^- \leq \tau < \beta$, the operation $inj_A^\tau : A_\tau \to A_\beta$, is the *canonical injection* defined on each $a \in A_\tau$ by $inj_A^\tau(a) = (\tau, a)$.
We let $Alg_{typ}(\Sigma)$ denote the class of all S-typed Σ algebras. $\square$

Given an S–typed Σ algebra A and any function type $(\sigma \to \tau) \in S$ we may write $a(b)$ as an abbreviation for $eval_A^{(\sigma\to\tau)}(a, b)$, for any $a \in A_{(\sigma\to\tau)}$ and $b \in A_\sigma$. We conclude this section with an example of a transfinite higher–order algebra constructed from the term algebra $T(\Sigma, X)$.

2.6 Example. Let $S \subseteq H(\mathcal{B})$ be a type structure over a type basis $\mathcal{B}$ and let Σ be an S-typed signature. Let $X = \langle X_\tau \mid \tau \in S \rangle$ be an S-indexed family of sets of variables. Clearly $T(\Sigma, X)$ is not an S-typed Σ algebra (e.g. $T(\Sigma, X)_{(\sigma\to\tau)} \not\subseteq [T(\Sigma, X)_\sigma \to T(\Sigma, X)_\tau]$). However, we can use $T(\Sigma, X)$ to construct an S-typed Σ algebra $\overline{T(\Sigma, X)}$, which we call the *collapse of* $T(\Sigma, X)$. Define the S-indexed family of carrier sets $\overline{T(\Sigma, X)} = \langle \overline{T(\Sigma, X)}_\tau \mid \tau \in S \rangle$ by

$$\overline{T(\Sigma, X)}_\tau = \langle \ \bar{t} \mid t \in T(\Sigma, X)_\tau \ \rangle,$$

where $\tau \in S$ and each $\bar{t}$ is defined inductively on the complexity of types as follows. For each basic type $\tau \in S$ and each term $t \in T(\Sigma, X)_\tau$, define $\bar{t} = t$. For each product type $(\sigma \times \tau) \in S$ and each term $t \in T(\Sigma, X)_{(\sigma\times\tau)}$, define

$$\bar{t} = (\overline{proj^1(t)}, \overline{proj^2(t)}).$$

For each function type $(\sigma \to \tau) \in S$ and each term $t \in T(\Sigma, X)_{(\sigma\to\tau)}$, define $\bar{t} : \overline{T(\Sigma, X)}_\sigma \to \overline{T(\Sigma, X)}_\tau$ on each $\overline{t_0} \in \overline{T(\Sigma, X)}_\sigma$ by

$$\bar{t}(\overline{t_0}) = \overline{eval^{(\sigma\to\tau)}(t, t_0)}.$$

For each limit type $\beta \in S$ and each term $t \in T(\Sigma, X)_\beta$ define

$$\bar{t} = \begin{cases} (\tau, \overline{t_0}), & \text{if there exists } \tau \in S \text{ with } \beta^- \leq \tau < \beta \text{ and } t_0 \in T(\Sigma, X)_\tau \\ & \text{such that } inj^\tau(t_0) = t; \\ (\beta, t), & \text{otherwise.} \end{cases}$$

The constants and algebraic operations of $\overline{T(\Sigma, X)}$ are defined as follows. For each type $\tau \in S$ and each constant symbol $c \in \Sigma_{\lambda,\tau}$, define $c_{\overline{T(\Sigma,X)}} = \bar{c}$. For each $w = \tau(1) \ldots \tau(n) \in S^+$, $\tau \in S$, each function symbol $f \in \Sigma_{w,\tau}$ and any $(t_1, \ldots, t_n) \in T(\Sigma, X)^w$, define

$$f_{\overline{T(\Sigma,X)}}(\overline{t_1}, \ldots, \overline{t_n}) = \overline{f(t_1, \ldots, t_n)}.$$

Using the fact that $\overline{t_1} = \overline{t_2}$ if, and only if, $t_1 = t_2$ it is easily verified that $\overline{T(\Sigma, X)}$ is a well defined S-typed Σ algebra. In the next section we shall see that $\overline{T(\Sigma, X)}$ arises as a special case of the *collapsing construction* which is used in the proof of Collapsing Theorem 3.6. □

3 Algebraic Constructions

From both the viewpoint of algebra and specification theory we are mainly concerned with the structure of transfinite higher–order algebras up to isomorphism. As we will show this structure can be characterised by a set of first–order sentences which we refer to as the *axioms of extensionality*. We investigate which basic algebraic constructions preserve or induce extensionality.

Let S be a type structure over a type basis $\mathcal{B}$, let Σ be an S-typed signature and let $X = \langle X_\tau \mid \tau \in S \rangle$ be an S-indexed family of disjoint infinite sets of variables. To avoid the problems associated with empty carrier sets we assume that Σ is non-void, i.e. a ground term exists for each type $\tau \in S$.

3.1 Definition. The set $\mathbf{Ext} = \mathbf{Ext}_\Sigma$ of *extensionality sentences* over Σ is the set of all Σ sentences of the form:
(i) for each product type $(\sigma \times \tau) \in S$ and variables $x, y \in X_{(\sigma\times\tau)}$

$$\forall x \; \forall y \; (proj^1(x) = proj^1(y) \;\wedge\; proj^2(x) = proj^2(y) \Rightarrow x = y);$$

(ii) for each function type $(\sigma \to \tau) \in S$ and variables $x, y \in X_{(\sigma\to\tau)}$, $z \in X_\sigma$

$$\forall x \; \forall y \; (\forall z \; (eval^{(\sigma\to\tau)}(x, z) = eval^{(\sigma\to\tau)}(y, z)) \Rightarrow x = y);$$

(iii) for each limit type $\beta \in S$ and each type $\tau \in S$ such that $\beta^- \leq \tau < \beta$, and $x, y \in X_\tau$

$$\forall x \; \forall y \; (inj^\tau(x) = inj^\tau(y) \;\Rightarrow\; x = y);$$

(iv) for each limit type $\beta \in S$ and variables $x \in X_\tau, y \in X_{\tau'}$ for types $\tau, \tau' \in S$ such that $\beta^- \leq \tau, \tau' < \beta$ and $\tau \neq \tau'$

$$\forall x \; \forall y \; (inj^\tau(x) \neq inj^{\tau'}(y)).$$

We say that a Σ algebra A is *extensional* if, and only if, $A \models \mathbf{Ext}$. We let $Alg_{Ext}(\Sigma)$ denote the class $Alg(\Sigma, \mathbf{Ext})$ of all extensional Σ algebras, and $Alg_{Ext}(\Sigma, \Phi)$ denote the class $Alg(\Sigma, \mathbf{Ext} \cup \Phi)$, for any set $\Phi \subseteq \mathcal{L}(\Sigma, X)$ of Σ formulas. □

Clearly, every S-typed Σ algebra is extensional. However, there exist extensional Σ algebras which are not S–typed Σ algebras, for example consider the term algebra $T(\Sigma, X)$. Nevertheless, given an extensional Σ algebra A we can always use the following *collapsing construction* to obtain an S–typed Σ algebra $\overline{A}$ which is isomorphic to A.

3.2 Definition. Let A be an extensional S-sorted Σ algebra. We define the *collapse* $\overline{A}$ of A to be the S-typed Σ algebra with S-indexed family of carrier sets $\overline{A} = \langle\ \overline{A}_\tau \mid \tau \in S\ \rangle$, where $\overline{A}_\tau = \{\overline{a} \mid a \in A_\tau\ \}$. We define each element $\overline{a} \in \overline{A}_\tau$ by induction on the complexity of types as follows. For each basic type $\tau \in S$ and each $a \in A_\tau$ define $\overline{a} = a$. For each product type $(\sigma \times \tau) \in S$ and each $a \in A_{(\sigma\times\tau)}$ define $\overline{a} = (\overline{proj^1_A(a)},\ \overline{proj^2_A(a)})$. For each function type $(\sigma \to \tau) \in S$ and each $a \in A_{(\sigma\to\tau)}$ define $\overline{a} : \overline{A}_\sigma \to \overline{A}_\tau$ on each $\overline{b} \in \overline{A}_\sigma$ by

$$\overline{a}(\overline{b}) = \overline{eval^{(\sigma\to\tau)}_A(a, b)}.$$

Finally, for each limit type $\beta \in S$ and each $a \in A_\beta$ define

$$\overline{a} = \begin{cases} (\tau, \overline{b}), & \text{if exists } \tau \in S,\ \beta^- \leq \tau < \beta \text{ and } b \in A_\tau \text{ such that } inj^\tau_A(b) = a; \\ (\beta, a), & \text{otherwise.} \end{cases}$$

The constants and algebraic operations of $\overline{A}$ are defined as follows. For each type $\tau \in S$ and each constant symbol $c \in \Sigma_{\lambda,\tau}$, define $c_{\overline{A}} = \overline{c_A}$. For each $\tau \in S$, each $w = \tau(1)\ldots\tau(n) \in S^+$, each function symbol $f \in \Sigma_{w,\tau}$ and any $(a_1, \ldots, a_n) \in A^w$, define $f_{\overline{A}}(\overline{a_1}, \ldots, \overline{a_n}) = \overline{f_A(a_1, \ldots, a_n)}$. □

In order to ensure that the operations of $\overline{A}$ are well defined we need the following result.

3.3 Proposition. *Let A be an extensional S-sorted Σ algebra. For each $\tau \in S$ and any $a, a' \in A_\tau$,*

$$\overline{a} = \overline{a'} \iff a = a'.$$

Proof. $\Leftarrow$ Trivial. $\Rightarrow$ By induction on the complexity of types. □

3.4 Corollary. *Let A be an extensional S-sorted Σ algebra. Then $\overline{A}$, the collapse of A, is a well defined S-typed Σ algebra.*

Proof. It can easily be shown that $\overline{A}$ is cumulative. Clearly, by Proposition 3.3 we know the operations of $\overline{A}$ are well defined. Thus, it only remains to show that the projection, evaluation and injection operation symbols are interpreted correctly in $\overline{A}$. This is straightforward and is left as an exercise for the reader. □

3.5 Lemma. *Let A be an extensional S-sorted Σ algebra, then $A \cong \overline{A}$.*

Proof. Using Proposition 3.3 it is easily verified that the map $\phi : A \to \overline{A}$ defined by $\phi_\tau(a) = \overline{a}$, for each $\tau \in S$ and $a \in A_\tau$, is a well defined isomorphism. □

An immediate consequence of the above Lemma is the following representation theorem for extensional Σ algebras which is a further generalisation of the Collapsing Theorem of Shepherdson and Mostowski.

3.6 Collapsing Theorem. (Mostowski, Shepherdson) *Let A be an S-sorted Σ algebra. Then A is isomorphic to an S-typed Σ algebra if, and only if, A is extensional.*

Proof. $\Rightarrow$ Follows since every S-typed Σ algebra is extensional.

$\Leftarrow$ Follows directly from Lemma 3.5. $\square$

We now consider which basic algebraic constructions preserve or induce extensionality. It can be easily shown that the class of all extensional Σ algebras need not be closed under the formation of subalgebras and this leads us to define a stronger notion of subalgebra.

3.7 Definition. Let A and B be Σ algebras. We say that A is an *extensional subalgebra* of B, written $A \leq_{Ext} B$ if, and only if, $A \leq B$ and $A \models \mathbf{Ext}$. Let K be any class of Σ algebras then K is *closed under extensional subalgebras* if, and only if, for any Σ algebras A and B, if $B \in K$ and $A \leq_{Ext} B$ then $A \in K$. $\square$

Clearly, $Alg_{Ext}(\Sigma)$ is closed under the formation of extensional Σ subalgebras. Since the extensionality sentences for any higher–order signature are all Horn sentences which are preserved under direct products it follows that the class of all extensional Σ algebras $Alg_{Ext}(\Sigma)$ is closed under non-empty direct products. Note that empty direct products are excluded since they correspond to unit algebras which are not in general extensional algebras. However, the class of all extensional Σ algebras need not be closed under homomorphic images and this leads us to define a more restrictive notion of homomorphic image.

3.8 Definition. Let A and B be Σ algebras. We say that B is an *extensional homomorphic image* of A if, and only if, B is a homomorphic image of A and B is extensional. For any class K of Σ algebras we say K is *closed under extensional homomorphic images* if, and only if, for any Σ algebras A and B, if $A \in K$ and B is an extensional homomorphic image of A then $B \in K$. $\square$

Clearly, the class of all extensional Σ algebras is closed under extensional homomorphic images. Next we introduce the notion of extensional Σ congruence.

3.9 Definition. Let $\equiv \, = \langle \equiv_\tau \subseteq A_\tau \times A_\tau \mid \tau \in S \rangle$ be a Σ congruence over an S-sorted Σ algebra A. Then $\equiv$ is said to be an *extensional* Σ *congruence* if, and only if, it satisfies the following four conditions.

(i) For each product type $(\sigma \times \tau) \in S$ and $a, a' \in A_{(\sigma\times\tau)}$,

$$proj_A^1(a) \equiv_\sigma proj_A^1(a') \text{ and } proj_A^2(a) \equiv_\tau proj_A^2(a') \implies a \equiv_{(\sigma\times\tau)} a'.$$

(ii) For each function type $(\sigma \to \tau) \in S$ and $a, a' \in A_{(\sigma\to\tau)}$,

$$\forall b \in A_\sigma \ (eval_A^{(\sigma\to\tau)}(a, b) \equiv_\tau eval_A^{(\sigma\to\tau)}(a', b)) \implies a \equiv_{(\sigma\to\tau)} a'.$$

(iii) For each limit type $\beta \in S$, each type $\tau \in S$, $\beta^- \leq \tau < \beta$ and any $a, a' \in A_\tau$,

$$inj_A^\tau(a) \equiv_\beta inj_A^\tau(a') \implies a \equiv_\tau a'.$$

(iv) For each limit type $\beta \in S$, any types $\tau, \tau' \in S$, $\beta^- \leq \tau, \tau' < \beta$ and $\tau \neq \tau'$,

$$inj_A^\tau(a) \not\equiv_\beta inj_A^{\tau'}(a'),$$

for all $a \in A_\tau$, $a' \in A_{\tau'}$. $\square$

Let A be an S-sorted Σ algebra then $Con_{Ext}(A)$ denotes the set of all extensional Σ congruences on A. Note that $Con_{Ext}(A)$ may be empty, for example if S contains a limit type with at least two types below it and A is a unit algebra. A class K of Σ algebras is said to be *closed under extensional quotients* if, and only if, for any Σ algebra A, if $A \in K$ and $\equiv$ is an extensional Σ congruence then $A/\equiv\, \in K$.

We can characterise those congruences on a Σ algebra A which yield extensional quotient algebras as follows.

3.10 Theorem. *Let A be an S-sorted Σ algebra and $\equiv$ be a Σ congruence on A. Then $A/\equiv$ is extensional if, and only if, $\equiv$ is an extensional Σ congruence on A.*

Proof. $\Rightarrow$ Suppose that $A/\equiv$ is extensional. Since the natural mapping $nat : A \to A/\equiv$ is an epimorphism it is straightforward to show that its kernel $\equiv$ is an extensional Σ congruence on A.

$\Leftarrow$ Suppose that $\equiv$ is extensional. Then we must prove that $A/\equiv$ is extensional. This is straightforward and is left as an exercise for the reader. $\Box$

4 Free Algebras

In this section we consider the existence of free and initial extensional algebras and the closure conditions necessary for a class of extensional Σ algebras to contain free and initial algebras.

Let Σ be an S-sorted signature and let K be a class of Σ algebras. Let F be a Σ algebra (not necessarily in K) and let $X \subseteq F$. Recall that F is said to be free for K on X if, and only if, for each $A \in K$ and each assignment $\alpha : X \to A$, there exists a unique homomorphic extension $\overline{\alpha} : F \to A$ of α. The S-indexed family X is termed the family of sets of generators for F and F is said to be freely generated by X. If in addition $F \in K$ then we say that F is free in K on X. If F is free for (respectively in) K on $\emptyset$ then there exists a unique homomorphism from F to each algebra $A \in K$ and we say that F is initial for (respectively in) K.

We now consider the existence and construction of free extensional algebras. Recall that for any S-typed signature Σ and any S-indexed family $X = \langle X_\tau \mid \tau \in S \rangle$ of sets of variables, $T(\Sigma, X)$ is free in $Alg(\Sigma)$ on X and is thus free for each class $K \subseteq Alg(\Sigma)$ on X. It is straigtforward to show that $T(\Sigma, X) \models \mathbf{Ext}$. Thus it follows that $T(\Sigma, X)$ is free in $Alg_{Ext}(\Sigma)$ on X and in particular, $T(\Sigma)$ is initial in $Alg_{Ext}(\Sigma)$. Recall the concrete construction of a free algebra for a non-empty class K of Σ algebras.

4.1 Definition. Let Σ be an S-typed signature and let K be any non-empty class of Σ algebras. Define the Σ congruence $\equiv^K = \langle \equiv^K_\tau \subseteq T(\Sigma, X)^2_\tau \mid \tau \in S \rangle$ for each $\tau \in S$ and any terms $t, t' \in T(\Sigma, X)_\tau$ by $t \equiv^K_\tau t'$ if, and only if, $\overline{\alpha}_\tau(t) = \overline{\alpha}_\tau(t')$ for each $A \in K$ and each assignment $\alpha : X \to A$, i.e. $K \models t = t'$. It can easily be shown that if K is a non–empty extensional class of Σ algebras then $\equiv^K$ is an extensional Σ congruence. $\Box$

We denote the quotient algebra $T(\Sigma, X)/\equiv^K$ by $T_K(\Sigma, X)$. By a basic result of universal algebra $T_K(\Sigma, X)$ is free for K on X. Furthermore, by a result of Birkhoff [1935] a sufficient condition for $T_K(\Sigma, X)$ to be in K is that K is closed under the formation of isomorphic images, direct products and subalgebras. However, as we have already seen in Section 3, the class $Alg_{Ext}(\Sigma)$ of all extensional Σ algebras need not be closed under subalgebras. So which extensional classes contain free algebras? To answer this question we generalise a result of Meinke [1992] which extended the result of Birkhoff to higher–order algebras with product and function types.

4.2 Theorem. *There exists an S-indexed family κ of cardinals such that for every S-indexed family X of sets of variables and every non-empty class K of extensional Σ algebras which is closed under isomorphic images, non-empty direct products and extensional subalgebras, if $|X| \geq \kappa$ then*

$$T_K(\Sigma, X) \in K.$$

Proof. Define the Σ congruence $\approx^K = \langle \approx_\tau^K \subseteq T(\Sigma, X)_\tau^2 \mid \tau \in S \rangle$ by $\approx^K = \bigcap C$, where

$$C = \{\equiv \in Con_{Ext}(T(\Sigma, X)) \mid T(\Sigma, X)/\equiv \text{ embeds in some } A \in K\}.$$

Then we can show there exists an S-indexed family κ of cardinals such that $\equiv^K$ and $\approx^K$ are identical extensional Σ congruences on $T(\Sigma, X)$ for any S-indexed family X of sets of variables with $|X| \geq \kappa$ (see Steggles [1995]). Let I be any indexing set for C such that $\approx^K = \bigcap_{i \in I} \equiv^i$ (note that I is non-empty since $\equiv^K = \approx^K$ is extensional and so cannot be the unit congruence). By a basic result of subdirect products, there exists a subdirect embedding

$$\epsilon : T(\Sigma, X)/\approx^K \to \Pi_{i \in I} T(\Sigma, X)/\equiv^i .$$

For each $i \in I$ we have $T(\Sigma, X)/\equiv^i$ embeds in some $A \in K$ so we can define a Σ embedding $\phi : T(\Sigma, X)/\equiv^i \to A$. By Theorem 3.10, $T(\Sigma, X)/\equiv^i$ is extensional and so $\phi(T(\Sigma, X)/\equiv^i) \leq_{Ext} A$. Furthermore $T(\Sigma, X)/\equiv^i \cong \phi(T(\Sigma, X)/\equiv^i)$. Therefore, since $A \in K$ and K is closed under isomorphic images and extensional subalgebras it follows that $T(\Sigma, X)/\equiv^i \in K$. Since K is closed under the formation of non-empty direct products it follows that $\Pi_{i \in I} T(\Sigma, X)/\equiv^i \in K$. As $\approx^K$ is an extensional Σ congruence it follows by Theorem 3.10 that $T(\Sigma, X)/\approx^K$ is extensional. Since ϵ embeds $T(\Sigma, X)/\approx^K$ in $\Pi_{i \in I} T(\Sigma, X)/\equiv^i$ and K is closed under the formation of isomorphic images and extensional subalgebras it follows that $T(\Sigma, X)/\approx^K \in K$. Thus, $T_K(\Sigma, X) \in K$. $\square$

Unfortunately, Theorem 4.2 provides no information about the existence of initial algebras. We know the class $Alg_{Ext}(\Sigma)$ of all extensional Σ algebras admits an initial algebra, namely $T(\Sigma)$, but this turns out to be unusual. Generally, most subclasses of $Alg_{Ext}(\Sigma)$, for example those satisfying some set of equations E, do not contain initial algebras. Nevertheless, a subclass $K \subseteq Alg_{Ext}(\Sigma)$ satisfying the closure conditions of Theorem 4.2, which admits a *minimal* algebra, does admit a minimal algebra which approximates the properties of an initial model. We refer to this minimal algebra as the (*transfinite*) *initial extensional model.* Given any class K of Σ algebras we let $Min(K)$ denote the class of all minimal Σ algebras in K.

4.3 Theorem. *Let K be any non-empty class of extensional Σ algebras which is closed under the formation of isomorphic images, non-empty direct products and extensional subalgebras. If K admits a minimal algebra then $T_{Min(K)}(\Sigma)$ is initial in $Min(K)$.*

Proof. Define the Σ congruence $\approx^{Min(K)} = \langle \approx_\tau^{Min(K)} \subseteq T(\Sigma)_\tau^2 \mid \tau \in S \rangle$ by

$$\approx^{Min(K)} = \bigcap \{\equiv \in Con_{Ext}(T(\Sigma)) \mid T(\Sigma)/\equiv \text{ embeds in some } A \in Min(K)\}.$$

Suppose K admits a minimal algebra. Then we can show that $\equiv^{Min(K)}$ and $\approx^{Min(K)}$ are identical extensional Σ congruences on $T(\Sigma)$ (see Steggles [1995]). The remainder of the proof is similar to the proof of Theorem 4.2. $\square$

We term $T_{Min(K)}(\Sigma)$ the (*transfinite*) *initial extensional model* to emphasize the fact that it is initial in a weaker, but still non-trivial, sense. For any equational theory $E \subseteq Eqn(\Sigma, X)$ we denote the initial extensional model in the class $Min_{Ext}(\Sigma, E)$ by $I_{Ext}(\Sigma, E)$.

5 Transfinite Higher–Order Equational Logic

We extend the higher–order equational logic presented in Meinke [1992] to a *higher–order equational logic with transfinite types* (referred to as simply *transfinite higher–order equational logic*). We show that transfinite higher–order equational logic is both sound and complete with respect to extensional Σ, E algebras. We then extend this transfinite higher–order equational logic with an infinitary inference rule for function types and show that this new calculus provides a concrete construction of the *initial extensional model* $I_{Ext}(\Sigma, E)$, which is initial in the class of all minimal extensional Σ, E algebras.

In the sequel let $S \subseteq H(\mathcal{B})$ be a type structure over some type basis $\mathcal{B}$ and let Σ be an S-typed signature. Let $X = \langle X_\tau \mid \tau \in S \rangle$ be an S-indexed family of non-empty sets of variable symbols. To avoid the problems associated with empty carrier sets in the many-sorted case we assume that Σ is non-void.

5.1 Definition. The (ordinary) inference rules of *transfinite higher–order equational logic* are the following.
(i) For any type $\tau \in S$ and any term $t \in T(\Sigma, X)_\tau$,

$$\frac{}{t = t}$$

is a *reflexivity* rule.
(ii) For any type $\tau \in S$ and any terms $t_0, t_1 \in T(\Sigma, X)_\tau$,

$$\frac{t_0 = t_1}{t_1 = t_0}$$

is a *symmetry* rule.
(iii) For any type $\tau \in S$ and any terms $t_0, t_1, t_2 \in T(\Sigma, X)_\tau$,

$$\frac{t_0 = t_1, \quad t_1 = t_2}{t_0 = t_2}$$

is a *transitivity* rule.
(iv) For each type $\sigma \in S$, any terms $t, t' \in T(\Sigma, X)_\sigma$, any type $\tau \in S$, any variable symbol $x \in X_\tau$ and any terms $t_0, t_1 \in T(\Sigma, X)_\tau$,

$$\frac{t = t', \quad t_0 = t_1}{t[x/t_0] = t'[x/t_1]}$$

is a *substitution* rule.
(v) For each product type $(\sigma \times \tau) \in S$ and any terms $t_0, t_1 \in T(\Sigma, X)_{(\sigma\times\tau)}$,

$$\frac{proj^{(\sigma\times\tau),1}(t_0) = proj^{(\sigma\times\tau),1}(t_1), \quad proj^{(\sigma\times\tau),2}(t_0) = proj^{(\sigma\times\tau),2}(t_1)}{t_0 = t_1}$$

is a *projection* rule.
(vi) For each function type $(\sigma \to \tau) \in S$, any terms $t_0, t_1 \in T(\Sigma, X)_{(\sigma\to\tau)}$ and any variable symbol $x \in X_\sigma$ not occurring in t_0 or t_1,

$$\frac{eval^{(\sigma\to\tau)}(t_0, x) = eval^{(\sigma\to\tau)}(t_1, x)}{t_0 = t_1}$$

is an *evaluation* rule.
(vii) For each limit type $\beta \in S$, each type $\tau \in S$ such that $\beta^- \leq \tau < \beta$ and any terms $t_0, t_1 \in T(\Sigma, X)_\tau$,

$$\frac{inj^\tau(t_0) = inj^\tau(t_1)}{t_0 = t_1}$$

is an *injection* rule. $\square$

Let $\vdash$ denote the inference relation between equational theories $E \subseteq Eqn(\Sigma, X)$ and equations $e \in Eqn(\Sigma, X)$, defined by $E \vdash e$ if, and only if, there exists a (finite) proof of e from E using the rules of transfinite higher–order equational logic. We can prove that $\vdash$ is *sound* with respect to extensional Σ, E algebras.

5.2 Soundness Theorem. *Let $E \subseteq Eqn(\Sigma, X)$ be an equational theory and let $e \in Eqn(\Sigma, X)$ be any equation. Then*

$$E \vdash e \Rightarrow Alg_{Ext}(\Sigma, E) \models e.$$

Proof. By induction on the length of proofs. $\square$

Note that an equational theory $E \subseteq Eqn(\Sigma, X)$ may have no extensional models, for example if it contains an equation which contradicts extensionality axiom 3.1.(iv). We say an equational theory $E \subseteq Eqn(\Sigma, X)$ is *consistent* if, and only if, $Alg_{Ext}(\Sigma, E) \neq \emptyset$.

Given an S-typed signature Σ and an equational theory $E \subseteq Eqn(\Sigma, X)$ we define the congruence $\equiv^E$ over the term algebra $T(\Sigma, X)$ by $t \equiv^E_\tau t' \Leftrightarrow E \vdash t = t'$, for $\tau \in S$ and any terms $t, t' \in T(\Sigma, X)_\tau$. Let $E \subseteq Eqn(\Sigma, X)$ be a consistent equational theory. Then it is straightforward to show that $\equiv^E$ is an extensional Σ congruence on $T(\Sigma, X)$. Thus, by Theorem 3.10 we know that $T(\Sigma, X)/\equiv^E$ is extensional.

Before being able to prove that $\vdash$ is complete with respect to extensional Σ, E algebras we need to show that $T(\Sigma, X)/\equiv^E$ is equationally generic.

5.3 Proposition. *Let $E \subseteq Eqn(\Sigma, X)$ be an equational theory and let $e \in Eqn(\Sigma, X)$ be any equation. Then*

$$E \vdash e \Leftrightarrow T(\Sigma, X)/\equiv^E \models e.$$

Proof. The proof is omitted for brevity (see Steggles [1995]). $\square$

We now prove that $\vdash$ is complete with respect to extensional Σ, E algebras.

5.4 Completeness Theorem. *Let $E \subseteq Eqn(\Sigma, X)$ be a consistent equational theory and let $e \in Eqn(\Sigma, X)$ be any equation. Then*

$$E \vdash e \Leftrightarrow Alg_{Ext}(\Sigma, E) \models e.$$

Proof. $\Rightarrow$ By Soundness Theorem 5.2.
$\Leftarrow$ Suppose that $E \not\vdash e$ then by Proposition 5.3 we know $T(\Sigma, X)/\equiv^E \not\models e$. By consistency $Alg_{Ext}(\Sigma, E) \neq \emptyset$ and so it follows that $T(\Sigma, X)/\equiv^E$ is extensional. Also by Proposition 5.3 we have $T(\Sigma, X)/\equiv^E \models E$ but $T(\Sigma, X)/\equiv^E \not\models e$. Thus, $Alg_{Ext}(\Sigma, E) \not\models e$. $\square$

In general the quotient algebra $T(\Sigma)/\equiv^E$ is not extensional and thus cannot be taken as the initial extensional model $I_{Ext}(\Sigma, E)$. To construct a quotient algebra of $T(\Sigma)$ which is both a model of E and extensional we need to add the following (infinitary) ω–evaluation rule to transfinite higher–order equational logic.

5.5 Definition. Let S be a type structure over a type basis $\mathcal{B}$. Let Σ be an S-typed signature. For each function type $(\sigma \to \tau) \in S$ and any terms $t_0, t_1 \in T(\Sigma, X)_{(\sigma \to \tau)}$,

$$\frac{\langle eval^{(\sigma \to \tau)}(t_0, t) = eval^{(\sigma \to \tau)}(t_1, t) \mid t \in T(\Sigma)_\sigma \rangle}{t_0 = t_1}$$

is an (infinitary) *ω–evaluation rule.* $\square$

Let $\vdash_\omega$ denote the inference relation between equational theories $E \subseteq Eqn(\Sigma, X)$ and equations $e \in Eqn(\Sigma, X)$, defined by $E \vdash_\omega e$ if, and only if, there exists a proof of e from E using the inference rules of transfinite higher–order equational logic and the ω-evaluation rule. Clearly, if $E \vdash_\omega e$ then $A \models e$ for every minimal extensional Σ, E algebra A, i.e. the ω-evaluation rule is *sound* with respect to minimal models. Given an equational theory $E \subseteq Eqn(\Sigma, X)$ we can use $\vdash_\omega$ to define a Σ congruence $\equiv^{E,\omega}$ on the ground term algebra $T(\Sigma)$ in the usual way. Let $Min_{Ext}(\Sigma, E)$ denote the class of all minimal, extensional Σ, E algebras. If $Min_{Ext}(\Sigma, E) \neq \emptyset$ then it is straightforward to show that $\equiv^{E,\omega}$ is an extensional Σ congruence on $T(\Sigma)$.

5.6 Proposition. *Let $E \subseteq Eqn(\Sigma, X)$ be an equational theory and let $e \in Eqn(\Sigma, X)$ be any equation. Then*

$$E \vdash_\omega e \Rightarrow T(\Sigma)/\equiv^{E,\omega} \models e.$$

Proof. The proof is omitted for brevity (see Steggles [1995]). $\square$

From the above proposition we know that $T(\Sigma)/\equiv^{E,\omega} \models E$. Thus if there exists a minimal extensional Σ, E algebra then by Theorem 3.10 and Proposition 5.6, $T(\Sigma)/\equiv^{E,\omega} \in Min_{Ext}(\Sigma, E)$.

Finally, we prove that $T(\Sigma)/\equiv^{E,\omega}$ is isomorphic to $T_{Min(K)}(\Sigma)$ and thus, that $T(\Sigma)/\equiv^{E,\omega}$ is initial in $Min_{Ext}(\Sigma, E)$. Therefore it may be taken as a concrete construction of the *initial extensional model* $I_{Ext}(\Sigma, E)$.

5.7 Theorem. *Let $E \subseteq Eqn(\Sigma, X)$ be any equational theory and let $K = Alg_{Ext}(\Sigma, E)$. If $Min_{Ext}(\Sigma, E) \neq \emptyset$ then*

$$T(\Sigma)/\equiv^{E,\omega} \cong T_{Min(K)}(\Sigma),$$

and hence $T(\Sigma)/\equiv^{E,\omega}$ is initial in $Min(K)$.

Proof. Since K is an equational class we know by Theorem 4.3 that $T_{Min(K)}(\Sigma)$ is initial in $Min(K)$. By Theorem 3.10 and Proposition 5.6 we have $T(\Sigma)/\equiv^{E,\omega} \in Min_{Ext}(\Sigma, E)$. Therefore, there exists a homomorphism $\phi : T_{Min(K)}(\Sigma) \to T(\Sigma)/\equiv^{E,\omega}$. Since $T_{Min(K)}(\Sigma)$ and $T(\Sigma)/\equiv^{E,\omega}$ are both minimal Σ, E algebras we need only show that there exists a homomorphism $\psi : T(\Sigma)/\equiv^{E,\omega} \to T_{Min(K)}(\Sigma)$. Define ψ by $\psi_\tau([t]_{E,\omega}) = [t]_{Min(K)}$, for each $\tau \in S$ and each term $t \in T(\Sigma)_\tau$. Then ψ is well defined since the rules of transfinite higher–order equational logic and the ω-evaluation rule are sound for every minimal extensional Σ, E algebra. It only remains to prove that ψ is a homomorphism which is straightforward to do. $\square$

6 A Simple Specification Example

In this section we demonstrate the use of transfinite types for modelling polymorphism and partiality by considering the specification of a simple polymorphic functional programming language $\mathcal{PR}^\omega$ which is based on the functional programming language $\mathcal{PR}$ of primitive recursive functions (see Thompson [1987]). In this specification example the first limit type ω is used to allow infinite schemes of functions to be replaced by polymorphic functions. These polymorphic functions are partial and return a value of type *undef* whenever they are undefined for a particular type. We present an $\omega + 1$–order equational specification of $\mathcal{PR}^\omega$ which we show is correct under higher–order initial algebra semantics. Our definitions and results generalise those for the algebra of primitive recursive functions over the natural numbers presented in Meinke [1994].

For the remainder of this section let S be a sort set such that $nat \in S$ and let Σ be an S–sorted signature such that $0 \in \Sigma_{\lambda,nat}$ and $succ \in \Sigma_{nat,nat}$. Let A be any Σ algebra such that $A_{nat} = \mathbf{N}$, $0_A = 0 \in \mathbf{N}$ and $succ_A(n) = n+1$, for any $n \in \mathbf{N}$, and let $Spec = (\Sigma, E)$ be a correct first–order equational specification of A under first–order initial algebra semantics, i.e. $I(Spec) \cong A$.

We use the first–order signature Σ to define an $\omega+1$–order signature $PR^\omega(\Sigma)$ which names all the basic functions and function building operations in our polymorphic functional programming language. We then use the Σ algebra A to construct a standard $PR^\omega(\Sigma)$ model $PR^\omega(A)$. Using the equational specification $Spec = (\Sigma, E)$, which by assumption is a correct specification of A under first–order initial algebra semantics, we construct a higher–order equational specification $PR^\omega(Spec) = (PR^\omega(\Sigma), PR^\omega(E))$ which we show correctly specifies $PR^\omega(A)$ under higher–order initial algebra semantics.

The following technical definition is needed in the sequel.

6.1 Definition. Let $\mathcal{B}$ be any type basis. For any types $\tau(1), \ldots, \tau(n) \in H(\mathcal{B})$, $n \geq 1$ we let $(\tau(1) \times \cdots \times \tau(n))$ denote the product type $(\tau(1) \times (\tau(2) \times \cdots \times \tau(n)))$.

For each non–empty word $w = \tau(1) \ldots \tau(n) \in \mathcal{B}^+$ define the type $\overline{w} \in H(\mathcal{B})$ by

$$\overline{w} = (\tau(1) \times \cdots \times \tau(n)).$$

$\square$

We begin by constructing a signature $PR^\omega(\Sigma)$ for our polymorphic functional programming language using the signature Σ as follows.

6.2 Definition. Let $\mathcal{B}(S) = S \cup \{undef\}$ be a type basis and let $S_{PR^\omega} \subseteq H(\mathcal{B}(S))$ be a type structure over the type basis $\mathcal{B}(S)$ defined by

$$S_{PR^\omega} = PT \ \cup \ \{\omega, (\omega \to \omega)\},$$

where $PT = \bigcup_{i \in \mathbf{N}} PT_i$, and $PT_0 = \mathcal{B}(S)$ and $PT_{n+1} = PT_n \cup \{(\sigma \times \tau) \mid \sigma, \tau \in PT_n\}$. Define the S_{PR^ω}–typed signature $PR^\omega(\Sigma)$ to be the smallest S_{PR^ω}–indexed family of sets satisfying the following conditions.

For each $w \in S^*$ and $\tau \in S$ we have $\Sigma_{w,\tau} \subseteq PR^\omega(\Sigma)_{w,\tau}$ and $error \in PR^\omega(\Sigma)_{\lambda,undef}$. We have $eval^{(\omega\to\omega)} \in PR^\omega(\Sigma)_{(\omega\to\omega)\ \omega,\omega}$, for each product type $(\sigma \times \tau) \in S_{PR^\omega}$,

$$proj^{(\sigma\times\tau),1} \in PR^\omega(\Sigma)_{(\sigma\times\tau),\sigma}, \quad proj^{(\sigma\times\tau),2} \in PR^\omega(\Sigma)_{(\sigma\times\tau),\tau},$$

$$\langle . \rangle^{(\sigma\times\tau)} \in PR^\omega(\Sigma)_{\sigma\ \tau,(\sigma\times\tau)},$$

and for each $\tau \in S_{PR^\omega}$ such that $\tau < \omega$, $inj^\tau \in PR^\omega(\Sigma)_{\tau,\omega}$. For each $\tau \in S$ and each constant symbol $c \in \Sigma_{\lambda,\tau}$, $c^\omega \in PR^\omega(\Sigma)_{\lambda,(\omega\to\omega)}$. For each $w \in S^+$, $\tau \in S$ and each function symbol $f \in \Sigma_{w,\tau}$, $f^\omega \in PR^\omega(\Sigma)_{\lambda,(\omega\to\omega)}$. We also have

$$id^\omega,\ und\_fn,\ proj^{\omega,1},\ proj^{\omega,2} \in PR^\omega(\Sigma)_{\lambda,(\omega\to\omega)},$$

$$\langle . \rangle^\omega, und? \in PR^\omega(\Sigma)_{\omega\ \omega,\omega}, \quad |, \circ, pr, switch^\perp \in PR^\omega(\Sigma)_{(\omega\to\omega)\ (\omega\to\omega),(\omega\to\omega)},$$

and for each type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$,

$$case^\tau \in PR^\omega(\Sigma)_{(\omega\to\omega)\ (\omega\to\omega),(\omega\to\omega)}.$$

$\square$

Given any S_{PR^ω}–typed $PR^\omega(\Sigma)$ algebra B, any type $(\tau(1) \times \cdots \times \tau(n)) \in S_{PR^\omega}$, $n \geq 1$ and any $x_i \in B_{\tau(i)}$, $1 \leq i \leq n$ we define $\langle x_1, \ldots, x_n \rangle_B$ by $\langle x_1 \rangle_B = x_1$ and for $n > 1$, $\langle x_1, \ldots, x_n \rangle_B = \langle x_1, \langle x_2, \ldots, x_n \rangle_B \rangle_B^{(\tau(1) \times \cdots \times \tau(n))}$.

We can construct a standard $PR^\omega(\Sigma)$ algebra $PR^\omega(A)$ using the Σ algebra A as follows.

6.3 Definition. We construct the S_{PR^ω}–typed $PR^\omega(\Sigma)$ algebra $PR^\omega(A)$ as the minimal subalgebra of a complete $\omega + 1$–order S_{PR^ω}–typed $PR^\omega(\Sigma)$ algebra P which is defined as follows.

Define the carrier sets of P by $P_{undef} = \{\perp\}$, for each type $\tau \in S$, $P_\tau = A_\tau$, for each product type $(\sigma \times \tau) \in S_{PR^\omega}$, $P_{(\sigma \times \tau)} = (P_\sigma \times P_\tau)$, and

$$P_\omega = \coprod_{\langle \tau \in S_{PR^\omega} | \tau < \omega \rangle} P_\tau, \quad P_{(\omega \to \omega)} = [P_\omega \to P_\omega].$$

Define the constants and operations of P as follows. For each $w \in S^*$, $\tau \in S$ and each symbol $f \in \Sigma_{w,\tau}$ define $f_P = f_A$ and define $error_P = \perp$. For each product type $(\sigma \times \tau) \in S_{PR^\omega}$ define $proj_P^{(\sigma \times \tau),1} : P_{(\sigma \times \tau)} \to P_\sigma$, $proj_P^{(\sigma \times \tau),2} : P_{(\sigma \times \tau)} \to P_\tau$ to be the left and right projection functions and define the pairing function $\langle . \rangle_P : P_\sigma \times P_\tau \to P_{(\sigma \times \tau)}$ on each $a \in P_\sigma$ and $b \in P_\tau$ by $\langle a, b \rangle_P = (a, b)$. Define $eval_P^{(\omega \to \omega)} : P_{(\omega \to \omega)} \times P_\omega \to P_\omega$ to be the evaluation function and for each type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$ define $inj_P^\tau : P_\tau \to P_\omega$ to be the canonical injection function. For each $\tau \in S$ and each constant symbol $c \in \Sigma_{\lambda,\tau}$ define the *constant function* $c_P^\omega : P_\omega \to P_\omega$ on each $a \in P_\omega$ by $c_P^\omega(a) = (\tau, c_A)$. For each $w \in S^+$, $\tau \in S$ and each function symbol $f \in \Sigma_{w,\tau}$ define $f_P^\omega : P_\omega \to P_\omega$ on each $a \in P_\omega$ by

$$f_P^\omega(a) = \begin{cases} (\tau, f_A(a_1, \ldots, a_n)), & \text{if } a = (\overline{w}, (a_1, \ldots, a_n)), \text{ for } (a_1, \ldots, a_n) \in P_{\overline{w}}; \\ (undef, \perp), & \text{otherwise.} \end{cases}$$

For each $a \in P_\omega$ define the *identity function* $id_P^\omega : P_\omega \to P_\omega$ by $id_P^\omega(a) = a$, the *undefined function* $und\_fn_P : P_\omega \to P_\omega$ by $und\_fn_P(a) = (undef, \perp)$, and the ω*–projection functions* $proj_P^{\omega,1}, proj_P^{\omega,2} : P_\omega \to P_\omega$ by

$$proj_P^{\omega,1}(a) = \begin{cases} (\sigma, a_1), & \text{if } a = ((\sigma \times \tau), (a_1, a_2)), \text{ for some product type} \\ & (\sigma \times \tau) \in S_{PR^\omega} \text{ and } (a_1, a_2) \in P_{(\sigma \times \tau)}; \\ (undef, \perp), & \text{otherwise;} \end{cases}$$

$$proj_P^{\omega,2}(a) = \begin{cases} (\tau, a_2), & \text{if } a = ((\sigma \times \tau), (a_1, a_2)), \text{ for some product type} \\ & (\sigma \times \tau) \in S_{PR^\omega} \text{ and } (a_1, a_2) \in P_{(\sigma \times \tau)}; \\ (undef, \perp), & \text{otherwise.} \end{cases}$$

For any $(\sigma, a), (\tau, b) \in P_\omega$ define the ω*–pairing function* $\langle . \rangle_P^\omega : P_\omega \times P_\omega \to P_\omega$ by

$$\langle (\sigma, a), (\tau, b) \rangle_P^\omega = ((\sigma \times \tau), (a, b)).$$

For any $a, b \in P_\omega$ define $und?_P : P_\omega \times P_\omega \to P_\omega$ by

$$und?_P(a, b) = \begin{cases} b, & \text{if } a = (undef, \perp); \\ a, & \text{otherwise.} \end{cases}$$

For any $f, g \in P_{(\omega \to \omega)}$ and any $a \in P_\omega$ define the *vectorisation function*

$$|_P : P_{(\omega \to \omega)} \times P_{(\omega \to \omega)} \to P_{(\omega \to \omega)}, \ (f|_P g)(a) = \langle f(a), g(a) \rangle_P^\omega,$$

and the *composition function*

$$\circ_P : P_{(\omega \to \omega)} \times P_{(\omega \to \omega)} \to P_{(\omega \to \omega)}, \ (f \circ_P g)(a) = f(g(a)).$$

We define the *primitive recursion operator* $pr_P : P_{(\omega\to\omega)} \times P_{(\omega\to\omega)} \to P_{(\omega\to\omega)}$ on any $f, g \in P_{(\omega\to\omega)}$ as follows. For each product type of the form $(nat \times \tau) \in S_{PR^\omega}$ and any $b \in P_\tau$ define

$$pr_P(f,g)((nat \times \tau),(0,b)) = f(\tau, b),$$

and for any $n \in P_{nat}$ define

$$pr_P(f,g)((nat \times \tau),(n+1,b)) =$$
$$g(\langle (nat,n), \langle (\tau,b), pr_P(f,g)((nat \times \tau),(n,b))\rangle_P^\omega \rangle_P^\omega).$$

For each $(\tau, b) \in P_\omega$ such that $\tau \neq (nat \times \sigma)$, for any $\sigma \in S_{PR^\omega}$ define

$$pr_P(f,g)(\tau,b) = (undef, \perp).$$

Define $switch_P^\perp : P_{(\omega\to\omega)} \times P_{(\omega\to\omega)} \to P_{(\omega\to\omega)}$ on each $a \in P_\omega$ by

$$switch_P^\perp(f,g)(a) = \begin{cases} g(a), & \text{if } f(a) = (undef, \perp); \\ f(a), & \text{otherwise.} \end{cases}$$

For each type $\tau \in S_{PR^\omega}$, $\tau < \omega$ define $case_P^\tau : P_{(\omega\to\omega)} \times P_{(\omega\to\omega)} \to P_{(\omega\to\omega)}$ on each $a \in P_\omega$ by

$$case_P^\tau(f,g)(a) = \begin{cases} f(a), & \text{if } a = (\tau, b), \text{ for some } b \in P_\tau; \\ g(a), & \text{otherwise.} \end{cases}$$

Define $PR^\omega(A)$ to be the minimal subalgebra of P. Note that $PR^\omega(A)$ is a countable algebra and that the Σ algebra A is preserved in $PR^\omega(A)$, i.e. $A \cong PR^\omega(A)|_\Sigma$. □

We now present some simple examples of polymorphic functions that can be constructed using the functional programming language $\mathcal{PR}^\omega$.

6.4 Examples.
(i) Consider the projection function $projn : PR^\omega(A)_\omega \to PR^\omega(A)_\omega$ defined on each $b \in PR^\omega(A)_\omega$ by

$$projn(b) = \begin{cases} (\tau_{i+1}, a_{i+1}), & \text{if } b = ((nat \times \tau(1) \times \cdots \times \tau(n)),(i, a_1, \ldots, a_n)), \\ & \text{for some } n \geq 1 \text{ and } i < n; \\ (undef, \perp), & \text{otherwise.} \end{cases}$$

Define the function term

$$PROJ \equiv (switch^\perp(proj^{\omega,1}, id^\omega) \circ (pr(id^\omega, (proj^{\omega,2} \circ (proj^{\omega,2} \circ proj^{\omega,2}))))).$$

Then we can show that $PROJ_{PR^\omega(A)} = projn$.
(ii) Let S_{NB} be a sort set such that $nat, bool \in S_{NB}$ and let Σ^{NB} be an S_{NB}–sorted signature such that

$$0 \in \Sigma^{NB}_{\lambda,nat},\ succ \in \Sigma^{NB}_{nat,nat},\ True, False \in \Sigma^{NB}_{\lambda,bool},\ EOR \in \Sigma^{NB}_{bool\ bool,bool}.$$

Let B be an S_{NB}–sorted Σ^{NB} algebra with carrier sets $B_{nat} = \mathbf{N}$, $B_{bool} = \mathbf{B}$, and constants and functions defined by $0_B = 0$, $True_B = tt$, $False_B = \mathit{ff}$, $succ_B(n) = n+1$, and

$$EOR_B(b_1, b_2) = \begin{cases} tt, & \text{if either } b_1 = tt \text{ or } b_2 = tt \text{ but not both;} \\ \mathit{ff}, & \text{otherwise;} \end{cases}$$

for any $n \in \mathbf{N}$ and $b_1, b_2 \in \mathbf{B}$.

Consider the addition function $add : PR^\omega(B)_\omega \to PR^\omega(B)_\omega$ defined on each $a \in PR^\omega(B)_\omega$ by

$$add(a) = \begin{cases} (nat, n+m), & \text{if } a = ((nat \times nat), (n, m)); \\ (undef, \perp), & \text{otherwise.} \end{cases}$$

Define the function term

$$ADD \equiv pr(case^{nat}(id^\omega, und\_fn), succ^\omega \circ proj^{\omega,2} \circ proj^{\omega,2}).$$

Then it is straightforward to show $ADD_{PR^\omega(B)} = add$. We can use $case^\tau_{PR^\omega(B)}$ and $switch^\perp_{PR^\omega(B)}$ to combine function terms to produce overloaded polymorphic functions. For example consider the function $add\_eor : PR^\omega(B)_\omega \to PR^\omega(B)_\omega$ defined on each $a \in PR^\omega(B)_\omega$ by

$$add\_eor(a) = \begin{cases} (nat, n+m), & \text{if } a = ((nat \times nat), (n, m)); \\ (bool, EOR_B(b_1, b_2)), & \text{if } a = ((bool \times bool), (b_1, b_2)); \\ (undef, \perp), & \text{otherwise.} \end{cases}$$

Then we can show that

$$case^{(nat \times nat)}(ADD, EOR^\omega)_{PR^\omega(B)} = add\_eor,$$

$$switch^\perp(ADD, EOR^\omega)_{PR^\omega(B)} = add\_eor.$$

□

We construct an $\omega+1$–order equational specification of $PR^\omega(A)$ based on the first–order equational specification $Spec = (\Sigma, E)$ for the Σ algebra A as follows.

6.5 Definition. Let X be an S_{PR^ω}–indexed family of sets of variables. Define the $\omega+1$–order equational specification $PR^\omega(Spec)$ by

$$PR^\omega(Spec) = (PR^\omega(\Sigma), PR^\omega(E)),$$

where $PR^\omega(E) \subseteq Eqn(PR^\omega(\Sigma), X)$ is an $\omega+1$–order equational theory consisting of the first–order equations E of $Spec$ and the following equations.

Let $Y, Z \in X_{(\omega \to \omega)}$, $x, y \in X_\omega$, $t \in X_{nat}$ and for each type $\tau \in S_{PR^\omega}$, $x^\tau, y^\tau \in X_\tau$. For each product type $(\sigma \times \tau) \in S_{PR^\omega}$ we have the equations

$$proj^1(\langle x^\sigma, y^\tau \rangle) = x^\sigma, \; proj^2(\langle x^\sigma, y^\tau \rangle) = y^\tau. \tag{1a, b}$$

For each $\tau \in S$ and each constant symbol $c \in \Sigma_{\lambda,\tau}$ we have the equation

$$c^\omega(x) = inj^\tau(c). \tag{2}$$

For each $w = \tau(1) \ldots \tau(n) \in S^+$, $\tau \in S$, each function symbol $f \in \Sigma_{w,\tau}$ and variables $x_1 \in X_{\tau(1)}, \ldots, x_n \in X_{\tau(n)}$ we have the equation

$$f^\omega(inj^{\overline{w}}(\langle x_1, \ldots, x_n \rangle)) = inj^\tau(f(x_1, \ldots, x_n)), \tag{3a}$$

and for each $\sigma \in S_{PR^\omega}$ such that $\sigma < \omega$ and $\sigma \neq \overline{w}$,

$$f^\omega(inj^\sigma(x^\sigma)) = inj^{undef}(error). \tag{3b}$$

We have the equations

$$id^\omega(x) = x, \tag{4}$$

$$und\_fn(x) = inj^{undef}(error). \tag{5}$$

For each product type $(\sigma \times \tau) \in S_{PR^\omega}$ we have the equations

$$proj^{\omega,1}(inj^{(\sigma\times\tau)}(\langle x^\sigma, y^\tau\rangle)) = inj^\sigma(x^\sigma), \tag{6a}$$

$$proj^{\omega,2}(inj^{(\sigma\times\tau)}(\langle x^\sigma, y^\tau\rangle)) = inj^\tau(y^\tau). \tag{6b}$$

For each basic type $\tau \in \mathcal{B}(S)$ we have the equations

$$proj^{\omega,1}(inj^\tau(x^\tau)) = inj^{undef}(error), \tag{6c}$$

$$proj^{\omega,2}(inj^\tau(x^\tau)) = inj^{undef}(error). \tag{6d}$$

For types $\sigma, \tau \in S_{PR^\omega}$ such that $\sigma, \tau < \omega$ we have the equation

$$\langle inj^\sigma(x^\sigma), inj^\tau(y^\tau)\rangle^\omega = inj^{(\sigma\times\tau)}(\langle x^\sigma, y^\tau\rangle). \tag{7}$$

For each type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$ and $\tau \neq undef$ we have the equations

$$und?(inj^{undef}(error), x) = x, \;\; und?(inj^\tau(y^\tau), x) = inj^\tau(y^\tau). \tag{8a, b}$$

We have the equations

$$(Y|Z)(x) = \langle Y(x), Z(x)\rangle^\omega, \tag{9}$$

$$(Y \circ Z)(x) = Y(Z(x)). \tag{10}$$

For each product type $(nat \times \tau) \in S_{PR^\omega}$ we have the equations

$$pr(Y, Z)(inj^{(nat\times\tau)}(\langle 0, x^\tau\rangle)) = Y(inj^\tau(x^\tau)), \tag{11a}$$

$$\begin{gathered} pr(Y, Z)(inj^{(nat\times\tau)}(\langle succ(t), x^\tau\rangle)) = \\ Z(\langle inj^{nat}(t), \langle inj^\tau(x^\tau), pr(Y, Z)(inj^{(nat\times\tau)}(\langle t, x^\tau\rangle))\rangle^\omega\rangle^\omega). \end{gathered} \tag{11b}$$

For each type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$ and $\tau \neq (nat \times \sigma)$, for any $\sigma \in S_{PR^\omega}$ we have the equation

$$pr(Y, Z)(inj^\tau(x^\tau)) = inj^{undef}(error). \tag{11c}$$

We have the equation

$$switch^\perp(Y, Z)(x) = und?(Y(x), Z(x)). \tag{12}$$

For each type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$ we have the equations

$$case^\tau(Y, Z)(inj^\tau(x^\tau)) = Y(inj^\tau(x^\tau)), \tag{13a}$$

and for each type $\sigma \in S_{PR^\omega}$ such that $\sigma < \omega$ and $\sigma \neq \tau$,

$$case^\tau(Y, Z)(inj^\sigma(x^\sigma)) = Z(inj^\sigma(x^\sigma)). \tag{13b}$$

□

We need to show that $PR^\omega(Spec)$ is a correct $\omega + 1$–order equational specification of $PR^\omega(A)$ under higher–order initial algebra semantics. Following the approach of Meinke [1994] we do this using a normalisation result for $PR^\omega(\Sigma)$ terms. We begin by showing that $PR^\omega(E)$ is a consistent equational theory.

6.6 Proposition. $PR^{\omega}(A) \models PR^{\omega}(E)$.

Proof. Clearly $PR^{\omega}(A) \models E$ since $A \cong PR^{\omega}(A)|_{\Sigma}$ and $A \models E$. Thus we need only show that $PR^{\omega}(A)$ is a model of the remaining equations of $PR^{\omega}(E)$. This is straightforward and is left as an exercise to the reader. $\Box$

Since *Spec* is a correct equational specification of A under first–order initial algebra semantics we know there must exist an isomorphism $\psi : A \to T(\Sigma)/\equiv^{E}$. We can decompose ψ such that $nat^{E} \circ \theta = \psi$, where $\theta : A \to T(\Sigma)$ may be assumed to be a homomorphism (since ψ is a homomorphism) and $nat^{E} : T(\Sigma) \to T(\Sigma)/\equiv^{E}$ is the natural mapping associated with the term congruence $\equiv^{E}$. We use the homomorphism θ to define a term representation for the first–order and ω–order elements of $PR^{\omega}(A)$.

6.7 Definition. Let $\theta : A \to T(\Sigma)$ be the homomorphism given above. For each type $\tau \in S_{PR^{\omega}}$ such that $\tau \neq (\omega \to \omega)$ define the mapping $\alpha(\theta)_{\tau} : PR^{\omega}(A)_{\tau} \to T(PR^{\omega}(\Sigma))_{\tau}$ by

$$\alpha(\theta)_{undef}(\perp) = error,$$

for any type $\tau \in S$ and $a \in PR^{\omega}(A)_{\tau}$

$$\alpha(\theta)_{\tau}(a) = \theta_{\tau}(a),$$

for any product type $(\sigma \times \tau) \in S_{PR^{\omega}}$ and any $(a, b) \in PR^{\omega}(A)_{(\sigma\times\tau)}$

$$\alpha(\theta)_{(\sigma\times\tau)}(a, b) = \langle \alpha(\theta)_{\sigma}(a), \alpha(\theta)_{\tau}(b) \rangle,$$

and for any type $\sigma \in S_{PR^{\omega}}$ such that $\sigma < \omega$ and $a \in PR^{\omega}(A)_{\sigma}$

$$\alpha(\theta)_{\omega}(\sigma, a) = inj^{\sigma}(\alpha(\theta)_{\sigma}(a)).$$

$\Box$

We now prove the following normalisation result.

6.8 Normalisation Lemma. *For any type $\tau \in S_{PR^{\omega}}$ such that $\tau \neq (\omega \to \omega)$ and any term $t \in T(PR^{\omega}(\Sigma))_{\tau}$,*

$$PR^{\omega}(E) \vdash t = \alpha(\theta)_{\tau}(t_{PR^{\omega}(A)}).$$

Proof. By induction on the complexity of terms.

Basis. Suppose $t \equiv error$ or $t \equiv c$, for some $\tau \in S$ and constant symbol $c \in \Sigma_{\lambda,\tau}$. Then by definition of $PR^{\omega}(A)$, $\alpha(\theta)$ and since θ is a homomorphism we have using reflexivity

$$PR^{\omega}(E) \vdash t = \alpha(\theta)_{\tau}(t_{PR^{\omega}(A)}).$$

Induction step. We need to show that for each $w = \tau(1) \ldots \tau(n) \in S^{+}_{PR^{\omega}}$, $\tau \in S_{PR^{\omega}}$ such that $\tau \neq (\omega \to \omega)$, $f \in PR^{\omega}(\Sigma)_{w,\tau}$ and any terms $t_1 \in T(PR^{\omega}(\Sigma))_{\tau(1)}, \ldots, t_n \in T(PR^{\omega}(\Sigma))_{\tau(n)}$ that

$$PR^{\omega}(E) \vdash f(t_1, \ldots, t_n) = \alpha(\theta)_{\tau}(f(t_1, \ldots, t_n)_{PR^{\omega}(A)}).$$

We prove this only for $\langle . \rangle \in PR^{\omega}(\Sigma)_{\sigma\ \tau,(\sigma\times\tau)}$, for each product type $(\sigma \times \tau) \in S_{PR^{\omega}}$, $und? \in PR^{\omega}(\Sigma)_{\omega\ \omega,\omega}$ and $eval^{(\omega\to\omega)} \in PR^{\omega}(\Sigma)_{(\omega\to\omega)\ \omega,\omega}$. We leave the proofs for the remaining four cases (which follow along similar lines) as an exercise for the reader.

Case (1) Suppose $t \equiv \langle t^1, t^2 \rangle$, for $(\sigma \times \tau) \in S_{PR^\omega}$ and $t^1 \in T(PR^\omega(\Sigma))_\sigma$, $t^2 \in T(PR^\omega(\Sigma))_\tau$. Then by the induction hypothesis we have

$$PR^\omega(E) \vdash \langle t^1, t^2 \rangle = \langle \alpha(\theta)_\sigma(t^1_{PR^\omega(A)}), \alpha(\theta)_\tau(t^2_{PR^\omega(A)}) \rangle. \tag{1}$$

Also by the definition of $\alpha(\theta)$ and $\langle . \rangle_{PR^\omega(A)}$

$$\alpha(\theta)_{(\sigma \times \tau)}(\langle t^1, t^2 \rangle_{PR^\omega(A)}) = \langle \alpha(\theta)_\sigma(t^1_{PR^\omega(A)}), \alpha(\theta)_\tau(t^2_{PR^\omega(A)}) \rangle. \tag{2}$$

So using (1) and (2) above we have

$$PR^\omega(E) \vdash \langle t^1, t^2 \rangle = \alpha(\theta)_{(\sigma \times \tau)}(\langle t^1, t^2 \rangle_{PR^\omega(A)}).$$

Case (2) Suppose $t \equiv und?(t^1, t^2)$, for some terms $t^1, t^2 \in T(PR^\omega(\Sigma))_\omega$. Then we have two possible cases to consider.

Subcase (1) Suppose $t^1_{PR^\omega(A)} = (undef, \perp)$. Then by the induction hypothesis and definition of $\alpha(\theta)$ we have

$$PR^\omega(E) \vdash und?(t^1, t^2) = und?(inj^{undef}(error), \alpha(\theta)_\omega(t^2_{PR^\omega(A)})). \tag{3}$$

By equation 6.5.(8a) and definition of $und?_{PR^\omega(A)}$ we have

$$\begin{aligned} PR^\omega(E) \vdash und?(inj^{undef}(error), \alpha(\theta)_\omega(t^2_{PR^\omega(A)})) = \\ \alpha(\theta)_\omega(und?(t^1, t^2)_{PR^\omega(A)}). \end{aligned} \tag{4}$$

Thus using transitivity, (3) and (4) above

$$PR^\omega(E) \vdash und?(t^1, t^2) = \alpha(\theta)_\omega(und?(t^1, t^2)_{PR^\omega(A)}).$$

Subcase (2) Suppose $t^1_{PR^\omega(A)} = (\tau, a)$, for some type $\tau \in S_{PR^\omega}$ such that $\tau < \omega$ and $\tau \neq undef$. Then the proof is similar to subcase (1) above except equation 6.5.(8b) is used instead of 6.5.(8a).

Case (3) Finally, suppose $t \equiv eval^{(\omega \to \omega)}(t^1, t^2)$, for terms $t^1 \in T(PR^\omega(\Sigma))_{(\omega \to \omega)}$ and $t^2 \in T(PR^\omega(\Sigma))_\omega$. By the induction hypothesis

$$PR^\omega(E) \vdash t^1(t^2) = t^1(\alpha(\theta)_\omega(t^2_{PR^\omega(A)})). \tag{5}$$

By induction on the complexity of t^1 we can prove (see Steggles [1995]) that

$$PR^\omega(E) \vdash t^1(\alpha(\theta)_\omega(t^2_{PR^\omega(A)})) = \alpha(\theta)_\omega(t^1(t^2)_{PR^\omega(A)}),$$

from which using transitivity and (5) above the result follows. $\square$

We may now prove that $PR^\omega(Spec)$ is a correct $\omega+1$–order equational specification of $PR^\omega(A)$ under higher–order initial algebra semantics.

6.9 Correctness Theorem. $I_{Ext}(PR^\omega(Spec)) \cong PR^\omega(A)$.

Proof. Since $PR^\omega(A)$ and $I_{Ext}(PR^\omega(Spec))$ are minimal, extensional $PR^\omega(\Sigma)$, $PR^\omega(E)$ algebras, and $I_{Ext}(PR^\omega(Spec))$ is initial in $Min_{Ext}(PR^\omega(Spec))$ it suffices to construct a homomorphism

$$\phi : PR^\omega(A) \to I_{Ext}(PR^\omega(Spec)).$$

For each type $\tau \in S_{PR^\omega}$ and each term $t \in T(PR^\omega(\Sigma))_\tau$, define $\phi_\tau(t_{PR^\omega(A)}) = nat^{PR^\omega(E),\omega}(t)$, where $nat^{PR^\omega(E),\omega} : T(PR^\omega(\Sigma)) \to T(PR^\omega(\Sigma))/\equiv^{PR^\omega(E),\omega}$ is the natural mapping associated with the term congruence $\equiv^{PR^\omega(E),\omega}$. We need to show that ϕ is well–defined on each element $a \in PR^\omega(A)_\tau$.

Consider any type $\tau \in S_{PR^\omega}$ such that $\tau \neq (\omega \to \omega)$ and any terms $t, t' \in T(PR^\omega(\Sigma))_\tau$. Suppose that $t_{PR^\omega(A)} = t'_{PR^\omega(A)}$. Then using the Normalisation Lemma 6.8 we can show $PR^\omega(E) \vdash t = t'$, and so by the definition of ϕ and $\equiv^{PR^\omega(E),\omega}$ we have $\phi_\tau(t_{PR^\omega(A)}) = \phi_\tau(t'_{PR^\omega(A)})$.

Consider the function type $(\omega \to \omega) \in S_{PR^\omega}$ and any $t, t' \in T(PR^\omega(\Sigma))_{(\omega\to\omega)}$. Suppose that $t_{PR^\omega(A)} = t'_{PR^\omega(A)}$. Then for any term $t'' \in T(PR^\omega(\Sigma))_\omega$ we know that $t(t'')_{PR^\omega(A)} = t'(t'')_{PR^\omega(A)}$ and so by the Normalisation Lemma 6.8,

$$PR^\omega(E) \vdash t(t'') = t'(t'').$$

Since t'' was arbitrarily chosen we have that $PR^\omega(E) \vdash_\omega t = t'$ by the ω–evaluation rule and so by the definition of ϕ and $\equiv^{PR^\omega(E),\omega}$ it follows that $\phi_{(\omega\to\omega)}(t_{PR^\omega(A)}) = \phi_{(\omega\to\omega)}(t'_{PR^\omega(A)})$.

Clearly ϕ is homomorphism since the valuation mapping $(.)_{PR^\omega(A)}$ and the natural mapping $nat^{PR^\omega(E),\omega}$ are homomorphisms. □

7 Conclusions

We have presented a theory of *higher–order algebra with transfinite types* which extends the simple binary product and function type system of higher–order algebra with *limit types.* The aim of this extension was to provide a higher–order algebraic framework in which the problematic phenomena of polymorphism and partiality could be naturally modelled. We demonstrated the use of transfinite types for modelling partial functions and both parametric and overloading polymorphism by considering the specification of a simple polymorphic functional programming language $\mathcal{PR}^\omega$. In this specification case study we used the first limit type ω to replace an infinite scheme of functions by a single polymorphic function. The problem of coping with partial functions was addressed by having a basic type *undef* for undefined values and defining partial functions to be polymorphic functions which return an element of type *undef* when they are undefined. This approach allows the undefined values to be partitioned from the "normal" data values and provides a very natural axiomatisation of partial functions (compare this approach to the so called *error algebra* approach, see for example Goguen [1978] and Goguen et al [1978]).

Transfinite types can be used to model Π–types (see Steggles [1995]) and in fact this provided one of the original motivations for considering a theory of transfinite types. We note that transfinite types provide a decidable typing discipline whereas Π–types are undecidable, since the equality of two typing (recursive) functions is undecidable. Thus there are strong practical reasons for considering the theory of transfinite types.

A range of algebraic methods exist in the literature for coping with polymorphism, partiality and error handling. For example *order–sorted algebra* (see Poigné [1991], Goguen and Meseguer [1992], Goguen and Diaconescu [1994] and for higher–order order–sorted algebra see Qian [1993]), *unified algebras* (see Mosses [1989, 1994]) and *equational type logic* (see Manca et al [1989, 1990]). However, our approach of using transfinite types is new. For a survey of the different approaches to algebraically modelling polymorphism and partiality we recommend Mosses [1993] and Poigné [1994]. For work on classical transfinite type theory we refer the interested reader to L'Abbé [1953], Andrews [1965] and Pottinger [1993].

The theory of higher–order algebra with transfinite types we have presented is still in its infancy and needs to be further developed and investigated. One area in particular that needs to be addressed is the provision of case studies demonstrating the use of the above algebraic techniques in practice. We note that a selection of specification case studies demonstrating some of the possible applications of transfinite types can be found in Steggles [1995]. Further work is also needed to investigate the possible uses of the higher limit types.

We are very grateful for the helpful advice and comments of K. Meinke and J. R. Hindley during the preparation of this paper. We also acknowledge the financial support of the Engineering and Physical Sciences Research Council and the HCM MeDiCiS project.

8 References

P. B. Andrews. *A Transfinite Type Theory with Type Variables.* Studies in Logic and the foundations of Mathematics, North Holland, 1965.

G. Birkhoff. On the Structure of Abstract Algebras. In: *Proceedings of the Cambridge Philosophical Society*, Vol. 31, 1935.

P.M. Cohn. *Universal Algebra.* Harper and Row, New York, 1965.

K.J. Devlin. *Fundamentals of Contemporary Set Theory.* Springer–Verlag, 1979.

J.A. Goguen. Abstract Errors for Abstract Data Types. In:E.J. Neuhold (ed), *Formal Descriptions of Programming Concepts*, p. 491–522, North-Holland, 1978.

J.A. Goguen, J. Thatcher and E. G. Wagner. An initial algebra approach to the specification, correctness and implementations of abstract data types. In: R. T. Yeh (ed), *Current Trends in Programming Methodology IV*, pages 80–149, Prentice–Hall, 1978.

J.A. Goguen and J. Meseguer. Order–sorted algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 105:217–273, 1992.

J.A. Goguen and R. Diaconescu. An Oxford survey of order-sorted algebra. In: *Mathematical Structures in Computer Science*, Vol. 4, pages 363-392, Cambridge University Press, 1994.

J.M. Henle. *An Outline of Set Theory.* Springer–Verlag, 1986.

P. Kosiuczenko and K. Meinke. On the Power of Higher–Order Algebraic Specification Methods. Report CSR 13–94, Department of Computer Science, University College of Swansea, 1994.

M. L'Abbé. Systems of Transfinite Types Involving λ–Conversion. *Journal of Symbolic Logic.* 18:209–224, 1953.

V. Manca, A. Salibra and G. Scollo. On the nature of TELLUS: a Typed Equational Logic Look over Uniform Specifications. *Proceedings of 1989 Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 379, pages 338–349, Springer–Verlag, 1989.

V. Manca, A. Salibra and G. Scollo. Equational Type Logic. *Theoretical Computer Science*, 77:131–159, 1990.

K. Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.

K. Meinke. A recursive second order initial algebra specification of primitive recursion. *Acta Informatica*, 31:329–340, 1994.

K. Meinke. A completeness theorem for the expressive power of higher–order algebraic specifications. Technical Report CSR–13–95, Department of Computer Science, University of Wales, Swansea, 1995.

K. Meinke and L.J. Steggles. Specification and Verification in Higher Order Algebra: A Case Study of Convolution. In: J. Heering, K. Meinke, B. Möller and T. Nipkow (eds), *Proc. of HOA '93: An International Workshop on Higher Order Algebra, Logic and Term Rewriting*, Lecture Notes in Computer Science 816, pages 189–222, Springer–Verlag, 1994.

K. Meinke and J.V. Tucker. Universal algebra. In: S. Abramsky, D. Gabbay and T.S.E. Maibaum, (eds) *Handbook of Logic in Computer Science*, Volume I, pages 189–412. Oxford University Press, Oxford, 1993.

B. Möller. Algebraic specifications with higher–order operators. In: L.G.L.T. Meertens (ed), *Program specification and transformation*, North–Holland, 1987.

B. Möller, A. Tarlecki and M. Wirsing. Algebraic specifications of reachable higher–order algebras. In: D. Sannella and A. Tarlecki (eds), *Recent Trends in Data Type Specification*, Lecture Notes in Computer Science 332, pages 154–169, Springer-Verlag, 1988.

P.D. Mosses. Unified Algebras and Action Semantics. In: B. Monien and R. Cori (eds), *Proc. of STACS 89 : Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 349. Springer-Verlag, 1989.

P.D. Mosses. The use of sorts in algebraic specifications. In: *Proceedings of 8th Workshop on Abstract Data Types and 3rd COMPASS Workshop*, Lecture Notes in Computer Science 655, pages 66–91, Springer–Verlag, 1993.

P.D. Mosses. Unified Algebras and Abstract Syntax. In: H. Ehrig and R. Orejas (eds), *Recent Trends in Data Type Specification*, Lecture Notes in Computer Science 785, pages 280–294, Springer–Verlag, 1994.

A. Poigné. Once more on Order–Sorted Algebra. In: *Mathematical Foundations of Computer Science 1991*, Lecture Notes in Computer Science 520, Springer–Verlag, 1991.

A. Poigné. Identity and Existence, and Types in Algebra. In: H. Ehrig and R. Orejas (eds), *Recent Trends in Data Type Specification*, Lecture Notes in Computer Science 785, pages 53–78, Springer–Verlag, 1994.

G. Pottinger. A Classical Type Theory with Transfinite Types. In: L. J. M. Claesen and M. J. C. Gordon (eds), *Higher Order Logic Theorem Proving and its Applications*, Elsevier Science Publishers (North–Holland), 1993.

Z. Qian. Higher–Order Order–Sorted Algebra. *Acta Informatica*, 30:569–607, 1993.

L.J. Steggles. *Extensions of Higher–Order Algebra: Fundamental Theory and Case Studies.* Ph. D. Thesis, Computer Science Department, University of Wales, Swansea, submitted 1995.

B.C. Thompson. *Mathematical theory of synchronous concurrent algorithms.* Ph. D. Thesis, School of Computer Studies, University of Leeds, 1987.

W. Wechler. *Universal Algebra for Computer Scientists.* EATCS Monographs on Theoretical Computer Science 25, Springer-Verlag, 1992.