On Bunched Polymorphism (Extended Abstract)

Matthew Collinson[1], David Pym[1], and Edmund Robinson[2]

[1] University of Bath, BA2 7AY, United Kingdom
[2] Queen Mary, University of London, E1 4NS, United Kingdom

Abstract. We describe a polymorphic extension of the substructural lambda calculus $\alpha\lambda$ associated with the logic of bunched implications. This extension is particularly novel in that both variables and type variables are treated substructurally, being maintained through a system of zoned, bunched contexts. Polymorphic universal quantifiers are introduced in both additive and multiplicative forms, and then metatheoretic properties, including subject-reduction and normalization, are established. A sound interpretation in a class of indexed category models is defined and the construction of a generic model is outlined, yielding completeness. A concrete realization of the categorical models is given using pairs of partial equivalence relations on the natural numbers. Polymorphic existential quantifiers are presented, together with some metatheory. Finally, potential applications to closures and memory-management are discussed.

1 Introduction

In recent years, substructural logics and type systems have become firmly established as fundamental tools in the analysis of programming languages. The most prominent are linear logics and types [5], but there are more ad hoc systems, designed for low-level languages and memory management, for example [18].

The logic of bunched implications, **BI**, as exposed in [10], [11], [13] is a substructural logic of growing importance. **BI** provides a logic of resource, which treats the sharing of resource, rather than the number of uses treated by linear logic. The resource-sensitive aspect of **BI** has led to it being adopted as the basis of the assertion language of new program logics, notably separation logic [15], which allow for safe-reasoning about imperative languages with pointers.

BI has several well-understood classes of models, both truth-functional and categorical, and like linear logics, has an elegant proof-theory. In particular, there is an associated lambda calculus, $\alpha\lambda$, giving a propositions-as-types correspondence. The calculus is presented using derivations of typing judgements in which contexts of typed variables are certain trees, called bunches. The way to understand $\alpha\lambda$ is through a reading of the terms known as the sharing interpretation which emphasizes the use of some computational resource. As an example of this, $\alpha\lambda$ has both additive and multiplicative function types. A function of the additive kind may make use of the same computational resource as its argument,

but this is not the case for the multiplicative. In [10], $\alpha\lambda$ was used to unify the Algol-like languages Syntactic Control of Interference (SCI) and Idealized Algol (IA), which had hitherto appeared to have irreconcilable features.

Whilst it has been demonstrated that **BI** has applications to program logic for imperative programming and to type systems for small, idealized languages, the full power of the type-system remains unexploited. The possibility exists to build a functional programming language along the lines of ML, but based on bunched rather than simple types. The typing of a program should then make guarantees about the use of resources (for example, memory, in the presence of references) as well as the compatibility of sub-expressions. This paper takes some of the first steps in that direction.

Polymorphism must be added to $\alpha\lambda$ in order to give a language with the expressivity of ML. We present a calculus which bears the same relationship to $\alpha\lambda$ as the Girard-Reynolds polymorphic lambda calculus $\lambda 2$ [3], [14] does to the simply-typed lambda calculus. Adding ordinary, impredicative polymorphism to $\alpha\lambda$ amounts to adding a further zone to typing contexts which manages the use of type variables. In this paper we take a further step, by considering a calculus in which the type variable zone consists of a bunch. This gives extra flexibility in the type system, for it allows us to consider both additive and multiplicative polymorphism. The additive polymorphism allows us to recover all standard uses of polymorphism, whilst the multiplicative polymorphism enforces non-sharing of resources associated with type variables. Multiplicative quantification closely resembles the freshness quantifier of Pitts and Gabbay [6]. Further steps and features are required before we have a genuinely ML-like type system, including predicative polymorphism, recursive types, references and typechecking.

In §2, we add polymorphic universal quantifiers to $\alpha\lambda$. We follow this with some of the more important metatheoretical results in §3. In §4, we describe an extension of the usual notion of categorical model. The additives are modelled in the usual way, and in a similar way, the multiplicatives are modelled by the right-adjoints to certain substitutions. In §5, we give an instance of such a model using the category *PER* of partial equivalence relations on the natural numbers.

In §6, we introduce polymorphic existential quantification. The desire to extend the sharing interpretation, together with metatheoretic concerns, governs the design of the multiplicative quantifier. The multiplicative existential is less semantically neat than the universal, but hints strongly at a number of applications, for it enables the hiding not just of a type, but also of the resources that accompany it. Thus there is an appealing intuition for multiplicative existentials as a kind of closure. We discuss connections to work on type systems for memory-management, specifically alias types [18] and regions [17], [19], where the use of location and region variables leads to forms of polymorphism. For alias types, this polymorphism appears to be multiplicative.

The work reported herein was carried out under the project 'Bunched ML', funded by the United Kingdom EPSRC. We acknowledge help and suggestions given by our collaborators, Josh Berdine and Peter O'Hearn of Queen Mary University London, during numerous discussions.

2 The Calculus

The calculus, which we shall call $\alpha 2\lambda 2$, has three levels of judgement. A first level judgement $X \vdash \tau$ gives a type τ over a bunch of type variables X. The second level, which has judgements of the form $X \vdash \Gamma$ generates the contexts Γ of ordinary variables over X. The third level comprises judgements $X \mid \Gamma \vdash M : \tau$ which show that a term M is well-typed with τ, given X and Γ.

Assume a countable collection of *type variables* $\alpha, \beta, \ldots$ to be given. The types used in the calculus are generated by

$$\tau := \top \mid I \mid \alpha \mid \tau \wedge \tau \mid \tau * \tau \mid \tau \to \tau \mid \tau \mathrel{-\!\!*} \tau \mid \forall \alpha.\tau \mid \forall_* \alpha.\tau \ ,$$

where α is any type variable. The connectives $\top$, $\wedge$, $\to$ and $\forall$ are the additive unit, product, function space and polymorphic universal quantifier, respectively. There are multiplicative unit I, product $*$, function space $\mathrel{-\!\!*}$ and universal $\forall_*$ connectives. We allow the letters σ, τ to range over types.

A *fibre (context)* is a bunch of type variables, generated as follows

$$X := \emptyset \mid \alpha \mid X, X \mid X; X \ ,$$

subject to the restriction that every type-variable may occur at most once in a bunch. Let X, Y, Z range over fibres.

Assume a countable collection of *variables* $x, y, z, \ldots$ to be given. A *(typing) context* is a bunch of typed variables, generated by

$$\Gamma := \emptyset \mid \emptyset_* \mid x : \tau \mid \Gamma, \Gamma \mid \Gamma; \Gamma \ ,$$

where x is a variable, τ is a type and any variable occurs at most once. The units $\emptyset$ and $\emptyset_*$ are distinct from the unit $\emptyset$ for fibres. The typing contexts are nothing more than the contexts of $\alpha\lambda$, but such that types may contain type variables.

Bunches are always subject to a pair of equivalence relations [13]. The first equivalence $\equiv$ on bunches is used to build structural rules that allow us to permute variables in fibres or contexts. It is given by commutative monoid rules for ";", for "," and by a congruence to ensure that the monoid rules can be applied at arbitrary depth in any bunch. The second relation $\cong$ is used to control contraction rules. The equivalence $\cong$ on fibres is simply renaming of type variables: $X \cong Y$ if Y can be obtained from X by renaming bijectively with type variables. The relation $\Gamma \cong \Delta$ between contexts holds just when Δ can be obtained by relabelling the variables of the leaves of Γ in a type preserving way: any leaf $x : \tau$ of Γ must correspond to a node $y : \tau$ of Δ.

There is an obvious notion of sub-bunch of a bunch. Let $B(B_1 \mid \ldots \mid B_n)$ be the notation for a bunch B with distinct, distinguished sub-bunches $B_1, \ldots, B_n$. Write $B[B_1'/B_1, \ldots B_n'/B_n]$ for the bunch formed by replacing each bunch B_i in B with B_i'.

The rules for generating *type formation judgements*, which specify types which are well-formed over fibres, are shown in Figure 1. A critical design decision is evident at this level. The formation rules for $\wedge$, $\to$, $*$ and $\mathrel{-\!\!*}$ are kept as simple as possible, in that formation takes place over a single, fixed fibre.

$$(TAx)\ \frac{}{\alpha \vdash \alpha} \qquad (T\top)\ \frac{}{\emptyset \vdash \top} \qquad \frac{}{\emptyset \vdash I}\ (TI)$$

$$(T\odot)\ \frac{X \vdash \sigma \quad X \vdash \tau}{X \vdash \sigma \odot \tau} \qquad (\odot \text{ is any of } \times, \to, *, -\!\!*)$$

$$(T\forall)\ \frac{X; \alpha \vdash \tau}{X \vdash \forall \alpha.\tau} \qquad \frac{X, \alpha \vdash \tau}{X \vdash \forall_* \alpha.\tau}\ (T\forall_*)$$

$$(TC)\ \frac{X(Y;Y') \vdash \tau}{X(Y) \vdash \tau[Y/Y']}\ (Y \cong Y') \qquad (TW)\ \frac{Y \vdash \tau}{X(Y) \vdash \tau} \qquad (Z \equiv Z')\ \frac{Z \vdash \tau}{Z' \vdash \tau}\ (TE)$$

Fig. 1. Type formation rules

The construction of contexts which are valid over fibres is generated from the type-formation judgements. These are presented as judgements of the form $X \vdash \Gamma$ where X is a fibre and Γ is a context and are characterised by: $X \vdash \Gamma$ holds if and only if $X \vdash \tau$ for each variable $x : \tau$ in Γ.

The terms of the language are given by the following grammar

$$\begin{array}{l} M := x \mid \top \mid I \mid \text{ let } I \text{ be } M \text{ in } M \\ \quad \mid \langle M, M \rangle \mid \pi_1 M \mid \pi_2 M \mid M * M \mid \text{ let } (x, y) \text{ be } M \text{ in } M \\ \quad \mid \lambda x : \tau.M \mid \mathrm{app}(M, M) \mid \lambda_* x : \tau.M \mid \mathrm{app}_*(M, M) \\ \quad \mid \Lambda \alpha.M \mid \mathrm{App}(M, X, \tau) \mid \Lambda_* \alpha.M \mid \mathrm{App}_*(M, X, \tau)\ , \end{array}$$

where α is a type variable, τ is a type, X is a fibre and x is a variable.

Let $FV(-)$ be the set of variables which are in a context $(-)$ or free (not bound by a lambda abstraction) in a term $(-)$. We use the notation $FTV(-)$ for the set of type variables which occur free in a bunch $(-)$, type $(-)$, the types of the variables in the context $(-)$ or the type of the term $(-)$, respectively. In a term $\mathrm{App}(M, X, \tau)$ or $\mathrm{App}_*(M, X, \tau)$, the type variables of X are free, so substitution must take account of this.

We introduce a syntactic measure μ which assigns to each term the set of type variables which are free and which occur in some application of the multiplicative universal quantifier. Formally, this is given by a recursive definition, where

$$\mu(\Lambda \alpha.M) = \mu(\Lambda_* \alpha.M) = \mu(M) \setminus \{\alpha\} \qquad \mu(\mathrm{App}_*(M, X, \tau)) = \mu(M) \cup FTV(X)$$

are the informative clauses.

The typing of terms uses the term and context formation judgements. The *term formation judgements* are derived according to a system of rules, a sample of which are shown in Figure 2. In addition to the rules shown, there are introduction and elimination rules for rules for additive $(\top)$ and multiplicative (I) units, additive $(\wedge)$ and multiplicative $(*)$ conjunction, additive lambda abstraction $(\to)$, contraction (C) and equivalence (E) for contexts. All of the rules

other than the quantifier rules and the fibre structurals use a fixed fibre X. That is to say, they are essentially the familiar rules for $\alpha\lambda$, but parameterised by the fibre. All the elimination rules, other than $(\forall E)$ and $(\forall_* E)$ are subject to the side-condition

$$\mu(N) \cap FTV(M) = \emptyset \qquad (\dagger)$$

which requires the separation of certain of the free type variables present.

$$(Ax)\ \frac{X \vdash x : \tau}{X \mid x : \tau \vdash x : \tau} \qquad \frac{X \mid \Gamma(\Delta) \vdash M : \tau \quad X \vdash \Delta'}{X \mid \Gamma(\Delta; \Delta') \vdash M : \tau}\,(W)$$

$$(\multimap I)\ \frac{X \mid \Gamma, x : \sigma \vdash M : \tau}{X \mid \Gamma \vdash \lambda_* x : \sigma . M : \sigma \multimap \tau} \qquad (\dagger)\frac{X \mid \Gamma \vdash N : \sigma \multimap \tau \quad X \mid \Delta \vdash M : \sigma}{X \mid \Gamma, \Delta \vdash \mathrm{app}_*(N, M) : \tau}\,(\multimap E)$$

$$(\forall I)\ \frac{X; \alpha \mid \Gamma \vdash M : \tau}{X \mid \Gamma \vdash \Lambda\alpha . M : \forall\alpha.\tau} \quad (\alpha \notin FTV(\Gamma)) \qquad \frac{X, \alpha \mid \Gamma \vdash M : \tau}{X \mid \Gamma \vdash \Lambda_*\alpha . M : \forall_*\alpha.\tau}\,(\forall_* I)$$

$$(\forall E)\ \frac{X \mid \Gamma \vdash M : \forall\alpha.\tau \quad Y \vdash \sigma}{X; Y \mid \Gamma \vdash \mathrm{App}(M, Y, \sigma) : \tau[\sigma/\alpha]} \qquad \frac{X \mid \Gamma \vdash M : \forall_*\alpha.\tau \quad Y \vdash \sigma}{X, Y \mid \Gamma \vdash \mathrm{App}_*(M, Y, \sigma) : \tau[\sigma/\alpha]}\,(\forall_* E)$$

$$(FW)\ \frac{Y \mid \Gamma \vdash M : \tau}{X(Y) \mid \Gamma \vdash M : \tau} \qquad (X \equiv Z)\ \frac{X \mid \Gamma \vdash M : \tau}{Z \mid \Gamma \vdash M : \tau}\,(FE)$$

$$(FC)\ \frac{X(Y; Y') \mid \Gamma \vdash M : \tau}{X(Y) \mid \Gamma[Y/Y'] \vdash M[Y/Y'] : \tau[Y/Y']}\,(Y \cong Y')$$

Fig. 2. Sample of the term formation rules

The usual rules for $\beta\eta\zeta$-conversions for $\alpha\lambda$ are retained, see [13]. In addition, we have four conversions for quantifiers,

$$\begin{array}{ll} \mathrm{App}(\Lambda\alpha.M, \alpha) \to_\beta M & \Lambda\alpha.\mathrm{App}(M, \alpha) : \tau \to_\eta M \\ \mathrm{App}_*(\Lambda_*\alpha.M, \alpha) \to_\beta M & \Lambda_*\alpha.\mathrm{App}_*(M, \alpha) : \tau \to_\eta M\ , \end{array}$$

where these terms are all typed over the same fibre X and context Γ such that α is not free in Γ. Let $\twoheadrightarrow$ be the reduction relation generated by the single step conversions. As usual, these relations give rise to a system of $\beta\eta\zeta$-equalities.

3 Metatheory

Many of the standard properties of a lambda calculus hold for $\alpha 2\lambda 2$.

Proposition 1. *(Substitution Laws)*

1. *If* $X \mid \Gamma(x:\sigma) \vdash N : \tau$ *and* $X \mid \Delta \vdash M : \sigma$ *are derivable and the condition* $\mu(N) \cap FTV(M) = \emptyset$ *holds then* $X \mid \Gamma[\Delta/x] \vdash N[M/x] : \tau$.
2. *If* $Y \mid \Gamma \vdash M : \tau$ *and* $Z \vdash \sigma$ *then* $Y[Z/\alpha] \mid \Gamma[\sigma/\alpha] \vdash M[(Z,\sigma)/\alpha] : \tau[\sigma/\alpha]$.

The side-condition on the first part is essential. This makes the side-condition (†) on the elimination laws necessary in order to prove subject-reduction.

Proposition 2. *The four rules below are admissible.*

$$\frac{X \mid \Gamma \vdash \lambda x:\sigma.M : \sigma \to \tau}{X \mid \Gamma; x:\sigma \vdash M : \tau} \qquad \frac{X \mid \Gamma \vdash \lambda_* x:\sigma.M : \sigma \multimap \tau}{X \mid \Gamma, x:\sigma \vdash M : \tau}$$

$$\frac{X \mid \Gamma \vdash \Lambda\alpha.M : \forall\alpha.\tau}{X;\alpha \mid \Gamma \vdash M : \tau} \qquad \frac{X \mid \Gamma \vdash \Lambda_*\alpha.M : \forall_*\alpha.\tau}{X,\alpha \mid \Gamma \vdash M : \tau}$$

The propositions above make it possible to prove subject-reduction.

Theorem 1. *If* $X \mid \Gamma \vdash M : \tau$ *and* $M \twoheadrightarrow N$ *then* $X \mid \Gamma \vdash N : \tau$ *is derivable.*

All reductions of the calculus terminate, as is shown by translation into the polymorphic lambda calculus $\lambda 2$.

Theorem 2. *The calculus is strongly normalizing.*

The reduction relation can be extended to include ζ-reductions (commuting conversions) for $*$, following [13], and the subject-reduction and normalization theorems continue to hold. Similarly, the extension of $\alpha 2\lambda 2$ with the additive disjunction $\vee$ of $\alpha\lambda$ causes no difficulties.

4 Categorical Semantics

We now give a categorical semantics to $\alpha 2\lambda 2$. This is a hybrid of the indexed category semantics of $\lambda 2$ with the doubly closed category semantics of $\alpha\lambda$.

Before giving the modified version of hyperdoctrine, we introduce some terminology for a certain structure on a category. Consider a symmetric monoid $(\otimes, I, a, l, r, s)$ on a category $\mathbb{B}$. Let $1_{\mathbb{B}} : \mathbb{B} \longrightarrow \mathbb{B}$ be the identity functor. The monoid $\otimes$ is a *pseudoproduct* if for every object B in $\mathbb{B}$ there is a (first) *pseudoprojection*, that is, a natural transformation $\psi^1_B : 1_{\mathbb{B}} \otimes B \Longrightarrow 1_{\mathbb{B}}$ satisfying the two coherence diagrams given below.

$$\begin{array}{ccc} (X \otimes Y) \otimes Z & \xrightarrow[a]{} & X \otimes (Y \otimes Z) \\ \downarrow{\scriptstyle \psi^1} & & \downarrow{\scriptstyle \psi^1} \\ X \otimes Y & \xrightarrow{\psi^1} & X \end{array} \qquad \begin{array}{ccc} X \otimes I & \xrightarrow[r]{} & X \\ \downarrow{\scriptstyle 1_{X\otimes I}} & & \downarrow{\scriptstyle 1_X} \\ X \otimes I & \xrightarrow{\psi^1} & X \end{array}$$

We write the component at an object A as $\psi^1_{A,B} : A \otimes B \longrightarrow A$. Using the symmetry isomorphisms s, it is easy to construct a second pseudoprojection

$\psi^2_A : A \otimes 1_{\mathbb{B}} \Longrightarrow 1_{\mathbb{B}}$ with components $\psi^2_{A,B}$ where A, B are any objects of $\mathbb{B}$. We frequently omit both subscripts and superscripts on pseudoprojections.

All products are pseudoproducts, but not vice versa. The category $Set_\perp$ of pointed sets $X_\perp$ and functions which preserve the distinguished element $\perp$ has a pseudoproduct given by the coproduct. A pseudoprojection from $X_\perp + Y_\perp$ to X may be taken to be $\perp, y \mapsto \perp,\ x \mapsto x$ for all $x \in X, y \in Y$.

A *cartesian doubly closed category* (CDCC) is a category with a pair of symmetric monoidal closed structures, one of which is cartesian. A functor between CDCC's is *strict* if it preserves both the cartesian closed and the monoidal closed structure on-the-nose. Let **CDCC** be the category of cartesian doubly closed categories and strict functors.

A *split indexed category* consists of a contravariant functor from a base category $\mathbb{B}$ to some category of categories, see [8] for a detailed account. A *hyperdoctrine* [16] is a categorical model of $\lambda 2$ consisting of a split indexed category with certain properties, including a system of adjunctions for modelling quantification. It also requires a distinguished base object Ω, called the *generic object*, which is characterized by the property that there is a natural bijection $\iota_J : (PJ)_0 \xrightarrow{\cong} \hom(J, \Omega)$, where $\hom(-, \Omega)$ is the contravariant hom functor for $\mathbb{B}$ and $(PJ)_0$ is the set of objects of the fibre PJ.

An $\alpha 2\lambda 2$-*hyperdoctrine* is a split indexed category $P : \mathbb{B}^{op} \longrightarrow \mathbf{CDCC}$ with: generic object; finite products and binary pseudoproducts in the base; for any projection π in the base, the functor $P(\pi)$ has a right-adjoint Π which satisfies the Beck-Chevalley condition; for any pseuduoprojection ψ in the base, the functor $P(\psi)$ has a right-adjoint Ψ which satisfies the Beck-Chevalley condition.

Interpret fibre-contexts X as objects in the base $\mathbb{B}$, with

$$[\![\emptyset]\!] = \top \qquad [\![\alpha]\!] = \Omega \qquad [\![X;Y]\!] = [\![X]\!] \times [\![Y]\!] \qquad [\![X,Y]\!] = [\![X]\!] \otimes [\![Y]\!] \ .$$

Interpret type formations as objects $[\![X \vdash \tau]\!]$ of the fibre $P([\![X]\!])$. An instructive fragment of the interpretation is given by

$$[\![\emptyset \vdash T : T]\!] = \top \qquad [\![\emptyset \vdash I : I]\!] = I \qquad [\![\alpha \vdash \alpha]\!] = \iota_\Omega^{-1}(1_\Omega)$$

$$[\![X \vdash \tau * \tau']\!] = [\![X \vdash \tau]\!] \otimes [\![X \vdash \tau']\!] \qquad [\![X \vdash \tau \multimap \tau']\!] = [\![X \vdash \tau]\!] \multimap [\![X \vdash \tau']\!]$$

$$[\![X \vdash \forall \alpha.\tau]\!] = \Pi[\![X; \alpha \vdash \tau]\!] \qquad [\![X \vdash \forall_* \alpha.\tau]\!] = \Psi[\![X, \alpha \vdash \tau]\!] \ ,$$

where $\top$, I, $\otimes$ and $\multimap$ are from the doubly closed structure of fibres, Π is adjoint to $\pi : [\![X]\!] \times \Omega \longrightarrow [\![X]\!]$, and Ψ is adjoint to $\psi : [\![X]\!] \otimes \Omega \longrightarrow [\![X]\!]$. The interpretations of the omitted rules are quite standard. In particular, the interpretation of the rule (TW) makes use of projections and pseudoprojections.

Interpret contexts as objects $[\![X \vdash \Gamma]\!]$ of $P([\![X]\!])$ by extension of the interpretation of types, using the product and monoidal structure of the fibre.

Morphisms $[\![X \mid \Gamma \vdash M : \tau]\!] : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \tau]\!]$ in $P([\![X]\!])$ are used to interpret term formations. A fragment of the interpretation is given below.

$$[\![X \mid x : \tau \vdash x : \tau]\!] = 1_{[\![X \vdash \tau]\!]} \qquad [\![X \mid \emptyset \vdash \top : \top]\!] = 1_{\top} \qquad [\![X \mid \emptyset_* \vdash I : I]\!] = 1_I$$

$$\frac{[\![X \mid \Gamma, x : \phi \vdash M : \psi]\!] = f : [\![X \vdash \Gamma]\!] \otimes [\![X \vdash \phi]\!] \longrightarrow [\![X \vdash \psi]\!]}{[\![X \mid \Gamma \vdash \lambda_* X : \phi.M : \phi \multimap \psi]\!] = f^{\frown} : [\![X \vdash \Gamma]\!] \longrightarrow ([\![X \vdash \phi]\!] \multimap [\![X \vdash \psi]\!])}$$

$$\frac{[\![X; \alpha \mid \Gamma \vdash M : \tau]\!] = g : [\![X; \alpha \vdash \Gamma]\!] \longrightarrow [\![X; \alpha \vdash \tau]\!]}{[\![X \mid \Gamma \vdash \Lambda\alpha.M : \forall\alpha.\tau]\!] = g^{\vartriangle} : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \forall\alpha.\tau]\!]}$$

$$\frac{[\![X, \alpha \mid \Gamma \vdash M : \tau]\!] = h : [\![X, \alpha \vdash \Gamma]\!] \longrightarrow [\![X, \alpha \vdash \tau]\!]}{[\![X \mid \Gamma \vdash \Lambda_*\alpha.M : \forall_*\alpha.\tau]\!] = h^{\triangleleft} : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \forall_*\alpha.\tau]\!]}$$

$$\frac{[\![X \mid \Gamma \vdash M : \forall\alpha.\tau]\!] = m : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \forall\alpha.\tau]\!] \qquad [\![Y \vdash \rho]\!] = B \in P([\![Y]\!])}{[\![X, Y \mid \Gamma \vdash \mathrm{App}(M, Y, \rho)]\!] = P(1_{[\![X]\!]} \times \iota(B))(m^{\triangledown}) : [\![X; Y \vdash \Gamma]\!] \longrightarrow [\![X; Y \vdash \tau[\rho/\alpha]]\!]}$$

$$\frac{[\![X \mid \Gamma \vdash M : \forall_*\alpha.\tau]\!] = m : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \forall_*\alpha.\tau]\!] \qquad [\![Y \vdash \rho]\!] = B \in P([\![Y]\!])}{[\![X, Y \mid \Gamma \vdash \mathrm{App}_*(M, Y, \rho)]\!] = P(1_{[\![X]\!]} \otimes \iota(B))(m^{\triangleright}) : [\![X, Y \vdash \Gamma]\!] \longrightarrow [\![X, Y \vdash \tau[\rho/\alpha]]\!]}$$

Here, $f^{\frown}$ is the linear exponential mate of f, $(-)^{\vartriangle}$ and $(-)^{\triangledown}$ give the transposes of $\pi^* \dashv \Pi$, $(-)^{\triangleleft}$ and $(-)^{\triangleright}$ give the transposes $\psi^* \dashv \Psi$, and $\iota(B) : B \longrightarrow \Omega$ is the base morphism arising from the fact that Ω is generic.

4.1 Soundness and Completeness

In any $\alpha 2\lambda 2$-hyperdoctrine, every judgement can be interpreted.

Proposition 3. *(Weak Soundness) Every judgement $X \mid \Gamma \vdash M : \tau$ has an interpretation as a morphism $[\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \tau]\!]$ in $P([\![X]\!])$.*

The interpretations of types and terms are coherent because of the splitting of the indexed category, the naturality and coherence conditions for the pseudoprojections and the Beck-Chevalley conditions.

Substitution of a term for a variable takes place in a fixed fibre, so its interpretation is modelled in that CDCC as in [13]. Interpreting substitution of types for type variables uses reindexing functors and the generic object.

Proposition 4. *(Equational Soundness) If $X \mid \Gamma \vdash M = M' : \tau$ is derivable then $[\![X \mid \Gamma \vdash M : \tau]\!] = [\![X \mid \Gamma \vdash M' : \tau]\!]$ holds.*

The syntactic equalities are generated by the $\beta\eta\zeta$-conversions. All of these take place over a fixed fibre, except for the reductions for the quantifiers. Over any fibre we have a CDCC and we know that the equalities over that fibre are all validated. The β- and η-rules for the multiplicative quantifier are witnessed, respectively, by the equations

$$(P(1_{[\![X]\!]} \otimes 1_{\Omega}))((m^{\triangleleft})^{\triangleright}) = m \qquad ((P(1_{[\![X]\!]} \otimes 1_{\Omega}))(n^{\triangleright}))^{\triangleleft} = n \ ,$$

given interpretations $[\![X, \alpha \mid \Gamma \vdash M : \tau]\!] = m : [\![X, \alpha \vdash \Gamma]\!] \longrightarrow [\![X, \alpha \vdash \tau]\!]$ and $[\![X \mid \Gamma \vdash N : \forall_*\alpha.\tau]\!] = n : [\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \forall_*\alpha.\tau]\!]$. These equalities

follow because the indexed category is split. The relevant equalities for additive quantification follow by the obvious modifications.

Completeness with respect to $\alpha 2\lambda 2$-hyperdoctrines is established by the usual method. That is, we build a generic model from the syntax such that if an equation holds between interpreted terms then it must also hold in the theory. The main novelty here is the construction of the base category, although this follows essentially the same pattern as the construction for $\lambda 2$ hyperdoctrines: objects are (bunches of) type variables and morphisms are substitutions derived from type formation judgements.

We construct the base $\mathbb{B}$ from the syntax of fibres and type formations. The objects of $\mathbb{B}$ are taken to be the equivalence classes of fibres under the congruence relation $\cong$, which handles α-conversion of type variables. Throughout this construction, we use fibres as representatives of equivalence classes. Let Ω be the equivalence class of α and $\top$ be the equivalence class of $\emptyset$.

The congruence $\cong$ on fibres extends to type formation judgements using substitution: $(X \vdash \tau) \cong (Y \vdash \tau') \iff (X \cong Y)\ \&\ (\tau' = \tau[Y/X])$ for all fibres X and Y. Again, we will tend to use representatives for equivalence classes in what follows. Define a mapping $\overline{(-)} : (X \vdash \tau) \mapsto \tau$ from type formations to types.

The morphisms of $\mathbb{B}$ from X to Y are certain trees with the same shape (internal node structure) as Y and with equivalence classes of type formations at the leaves. These morphisms are generated by an inductive definition.

There are a number of parts to the base case. These are identity, terminal, diagonal, projection, pseudoprojection, right unit, associativity, associativity inverse, symmetry. For brevity, we give only the diagonal and pseudoprojection clauses below. From these, the forms of the other cases may be easily inferred. Diagonal: for every X there is a morphism $\Delta_X : X \longrightarrow X; X'$, where X' is any fibre which is disjoint from X and with $X' \cong X$. The morphism is given by $f_X; f_{X'}$ where f_X is formed by replacing every leaf α of X with $X \vdash \alpha$. Pseudoprojection: for all X and Y there is an arrow $\psi^1 : X, Y \longrightarrow X$ formed by replacing each leaf α of X with $X, Y \vdash \alpha$.

The inductive definition has three step cases: product, pseudoproduct and composite. Product: if there are arrows $f : X \longrightarrow Y$ and $g : X \longrightarrow Y'$ then there is a morphism $X; X' \xrightarrow{f;g} Y; Y'$. It is formed as the tree $f'; g'$ where f' is formed from f by replacing each leaf $X \vdash \tau$ with $X; X' \vdash \tau$, and similarly for g'. Pseudoproduct: If there is a morphism $f : X \longrightarrow Y$ and there is a morphism $g : X' \longrightarrow Y'$ then there is a morphism $X, X' \xrightarrow{f,g} Y, Y'$. It is formed as the tree f', g' where f' is formed from f by replacing each leaf $X \vdash \tau$ with $X, X' \vdash \tau$, and similarly for g'. Composite: the composite in $\mathbb{B}$ of a pair of arrows $f : X \longrightarrow Y$, $g : Y \longrightarrow Z$ is an arrow $g \circ f : X \longrightarrow Z$ constructed by replacing each leaf $Y \vdash \rho$ of g with the leaf $X \vdash \rho[\overline{f}/Y]$, where the mapping $\overline{(-)}$ is extended to trees in the obvious way.

Some comments and observations about the above definition are in order. In a number of the clauses above we have formed a morphism from X to Y using some words like "replace any variable α of Y with the judgement $X \vdash \tau$" and it is to be understood that any leaves of Y which are units $\emptyset$ should be replaced

by the judgement $X \vdash \top$. Composition is a well-defined operation, independent of choices of representatives. The hom-sets of $\mathbb{B}$ are guaranteed to remain small.

It is a matter of lengthy calculation to verify that $\mathbb{B}$ is a category, has finite limits and has a symmetric monoid which is a pseudoproduct. These structures are suggested by the notation in the recursive definition.

Write $P(X)$ for the fibre over the equivalence class of X. The construction of each $P(X)$ follows the construction of a CDCC from $\alpha\lambda$, see [13]. *Objects* are equivalence classes of type formations $X \vdash \tau$, represented by pairs (X, τ). A *morphism* from (X, σ) to (X, τ) is an equivalence class of term formations $X \mid x : \sigma \vdash M : \tau$, where the equivalence is generated by α-equality for variables, the $\beta\eta\zeta$-rules (without the quantifier cases) and the congruence extended from the congruence $\cong$ on fibres.

Every arrow $u : X \longrightarrow Y$ of $\mathbb{B}$ yields a functor $P(u) : P(Y) \longrightarrow P(X)$ between fibres. The functor acts as $P(u)(Y, \tau) = (X, \tau[\overline{u}/Y])$ on any object (Y, τ) in $P(Y)$. The arrow assignment is given by $P(u)(Y, x, M) = (X, x, M[\overline{u}/Y])$ for any arrow (Y, x, M) in $P(Y)$. Both the object and arrow assignments can be verified to be well-defined and calculuations can be performed to show that $P(u)$ is indeed functorial.

Further calculations show that the functors $P(u)$ preserve the CDCC structure on-the-nose. Moreover, the functors induced by projections $\pi : X \times \Omega \longrightarrow X$ and pseudoprojections $\psi : X \otimes \Omega \longrightarrow X$ can be shown to have right-adjoints which satisfy the Beck-Chevalley conditions. The identity gives a natural bijection between the hom-sets $\mathbb{B}(X, \Omega)$ and fibres $P(X)$.

Theorem 3. *The functor* $P : \mathbb{B}^{op} \longrightarrow$ **CDCC** *is an* $\alpha 2\lambda 2$*-hyperdoctrine.*

The completeness theorem follows as a corollary, since P is constructed from the syntax and each term is interpreted, essentially, by itself.

Corollary 1. *(Completeness) If* $[\![X \mid \Gamma \vdash M : \tau]\!] = [\![X \mid \Gamma \vdash M' : \tau]\!]$ *holds in every* $\alpha 2\lambda 2$*-hyperdoctrine then* $X \mid \Gamma \vdash M = M' : \tau$ *is derivable in the calculus.*

5 A PER Model

Partial equivalence relations on the natural numbers give rise to one of the simplest and most elegant models of the polymorphic lambda calculus [4]. We show how to produce a PER model for $\alpha 2\lambda 2$.

A *partial equivalence relation*, PER for short, consists of a binary relation $R \subseteq \mathbb{N} \times \mathbb{N}$ on the natural numbers. Define $dom(R) = \{n \in \mathbb{N} \mid nRn\}$, the *domain* of R. A map between PERs consists of an equivalence class of codes for recursive functions that track from the source PER to the target PER, that is, functions which preserve the relation. Let *PER* be the category of partial equivalence relations and PER_0 be its set of objects. The category is cartesian closed. It also has binary coproducts: embed isomorphically the two given PERs into PERs with disjoint domains, then take the union of the relations.

Since *PER* is cartesian closed and has a symmetric monoid (given by the coproduct) we might think that we can use these two structures to model $\alpha\lambda$. However, the monoid fails to be closed. This can be remedied by moving to a model based on pairs of PERs, motivated by a similar construction for sets. The category $Set \times Set$ of pairs of sets is a CDCC, see [11], [13]. Finite products and exponentials are given pointwise. Moreover, there is an additional symmetric monoidal closed structure with

$$(A^0, A^1) \otimes (B^0, B^1) = ((A^0 \times B^0) + (A^1 \times B^1), (A^0 \times B^1) + (A^1 \times B^0))$$
$$(A^0, A^1) \multimap (B^0, B^1) = ((A^0 \to B^0) \times (A^1 \to B^1), (A^0 \to B^1) \times (A^1 \to B^0)) \ ,$$

for all $A^0, A^1, B^0, B^1 \in Set$, where $A + B$ is the coproduct of A and B in Set. This can be viewed as an instance $Set^{\mathbf{2}}$ of Day's closure construction [1], [2], where $\mathbf{2} = \{0, 1\}$ is the discrete category with monoid given by addition modulo two. Now $PER \times PER$ can be viewed as $PER^{\mathbf{2}}$ and so is doubly closed by [1]. Its operations are defined in the same way as those of $Set \times Set$, remembering that the $+$ in the definition of $\otimes$ is now the coproduct in PER. For any pair (A^0, A^1) of PERs let $(A^0, A^1)^0 = A^0$ and $(A^0, A^1)^1 = A^1$. Extend the notion of domain to pairs of PERS with $dom(A, B) = dom(A) \times dom(B)$ for any PERs A and B. For any function $f : A \longrightarrow B$ and $C \subseteq A$ let $f\restriction_C$ be the restriction of f to C.

Let X be a bunch of type variables. Let $dom(\rho) = \bigcup_{\alpha \in FTV(X)} \rho(\alpha)$ for any function $\rho : FTV(X) \longrightarrow PER_0 \times PER_0$. An *environment* for X is a function $\rho : FTV(X) \longrightarrow PER_0 \times PER_0$ such that if any (Y, Z) is a sub-bunch of X then $dom(\rho\restriction_Y) \cap dom(\rho\restriction_Z) = \emptyset$ holds. Let $Env(X)$ be the set of environments for X.

A *semantic type* (over X) is a function $\tau : Env(X) \longrightarrow PER_0 \times PER_0$ from environments to pairs of PERs. These definitions give a natural generalization of the ordinary PER model of polymorphism, in which an environment consists of a tuple of PERs and a semantic type consists of a map from environments to PERs. A map from τ to τ' (over X) is an equivalence class $[e]$ of codes for pairs of codes, $([e^0], [e^1])$, where the recursive function corresponding to each e^i tracks from $(\tau\rho)^i$ to $(\tau'\rho)^i$ for all environments ρ. This gives a category $P(X)$ of semantic types over X.

Let $\alpha_1, \ldots, \alpha_n$ be the type variables in X. A *subsitution* $(-)[\tau_1/\alpha_1, \ldots \tau_n/\alpha_n]$ for X consists of semantic types $\tau_1, \ldots, \tau_n$ over some bunch Y such that: if X has a sub-bunch (W, Z), where W has type variables with $\alpha_{i_1}, \ldots, \alpha_{i_p}$ and Z has type variables with $\alpha_{j_1}, \ldots, \alpha_{j_q}$ then $(-)[\tau_{i_1}/\alpha_{i_1}, \ldots, \tau_{i_p}/\alpha_{i_p}, \tau_{j_1}/\alpha_{j_1}, \ldots, \tau_{j_q}/\alpha_{j_q}]$ is a substitution for (W, Z) if $(-)[\tau_{i_1}/\alpha_{i_1}, \ldots \tau_{i_p}/\alpha_{i_p}]$ is a substitution for W and $(-)[\tau_{j_1}/\alpha_{j_1}, \ldots \tau_{j_q}/\alpha_{j_q}]$ is a substitution for Z and $dom(\tau_{i_l}(\rho)) \cap dom(\tau_{j_m}(\rho)) = \emptyset$ for all $\rho \in Env(Y)$ and $1 \leq l \leq p$ and $1 \leq m \leq q$. A map from Y to X is just such a substitution. This gives a category Bun of bunches of type variables.

If ρ is an environment for X and $A \in PER_0 \times PER_0$ then define a function $\rho^A : FTV(X) \cup \{\alpha\} \longrightarrow PER_0 \times PER_0$ by $\alpha \mapsto A$ and $\beta \mapsto \rho(\beta)$ for $\beta \neq \alpha$. Now ρ^A is an environment for $X; \alpha$. If A satisfies $A \cap dom(\rho) = \emptyset$ then ρ^A is also an environment for X, α.

Define semantic types τ_{α_i} over X, α by $\tau_{\alpha_i}(\rho) = \rho(\alpha_i)$, for each $1 \leq i \leq n$. Now $(-)[\tau_{\alpha_1}/\alpha_1, \ldots, \tau_{\alpha_n}/\alpha_n]$ defines a map from $X; \alpha$ to X, called π, and

also a map from X, α to X, called ψ. Each of these induces a functor, with $P(\pi)(\tau)(\rho) = \tau(\rho\restriction_{FTV(X)})$ for $\rho \in Env(X;\alpha)$ and $P(\psi)(\tau)(\rho) = \tau(\rho\restriction_{FTV(X)})$ for $\rho \in Env(X,\alpha)$, respectively.

If τ' is a semantic type over $X;\alpha$ or, respectively, X,α then

$$(\Pi\tau')\rho = \bigcap_{A \in PER_0 \times PER_0} \tau'(\rho^A) \qquad (\Psi\tau')\rho = \bigcap_{\substack{A \in PER_0 \times PER_0 \\ A \cap dom(\rho) = \emptyset}} \tau'(\rho^A) \ ,$$

for each environment ρ for X, defines a semantic type over X.

Let τ be a semantic type over X and τ' be a semantic type over $X;\alpha$ or X,α respectively. In the first case, a map from τ to $\Pi(\tau')$ is precisely the same thing as a map from $P(\pi)(\tau)$ to τ'. In the second case, a map from τ to $\Psi(\tau')$ is precisely the same thing as a map from $P(\psi)(\tau)$ to τ'. We therefore have natural bijections between arrows

$$\frac{\tau \longrightarrow \Pi(\tau')}{P(\pi)(\tau) \longrightarrow \tau'} \qquad \frac{\tau \longrightarrow \Psi(\tau')}{P(\psi)(\tau) \longrightarrow \tau'} \ ,$$

given by identity maps.

The above model is not quite a categorical model as described in the previous section. We produce a $\alpha 2\lambda 2$-hyperdoctrine by taking a suitable quotient on bunches to make the interpretation of all type variables identical.

6 Existential Quantifiers

Existential quantifiers may be defined in the polymorphic lambda calculus $\lambda 2$ and are closely connected to the concept of abstract data type [9]. In this section, we describe existential quantification in the bunched polymorphic setting, leading to both additive and multiplicative existentials.

First-order additive and multiplicative existential quantifiers have been studied in [11], [13]. Proof-theoretic considerations drive the design of the polymorphic existentials, just as they do in the first-order case.

Additive existential quantification, $\exists$, is quite straightforward to add to the system $\alpha 2\lambda 2$. However, the multiplicative quantifier, $\exists_*$, is very delicate. In particular, it requires a number of side-conditions which can interfere with the side-condition (†) used for $\alpha 2\lambda 2$. Rather than describing such a system in its full complexity, we first remove the universal quantifiers and instances of (†) before adding the existentials. However, in general, both universals and existentials can be considered together.

The grammars generating types and terms are extended with

$$\begin{array}{rl} \tau & ::= \ldots \mid \exists\alpha.\tau \mid \exists_*\alpha.\tau \\ M & ::= \ldots \mid \langle \phi, M \rangle \mid \textbf{unpack } M \textbf{ as } \langle\, \alpha, x\rangle \textbf{ in } M \\ & \quad \mid \langle Y, \phi, M \rangle_* \mid \textbf{unpack}_* \ M \textbf{ as } \langle \alpha, x \rangle \textbf{ in } M \ , \end{array}$$

where α and x are bound in **unpack** and $\textbf{unpack}_*$ terms.

Just as with the multiplicative universal quantifier, we are forced to use an additional syntactic measure with the multiplicative existential. The set $WR(M)$ of *witnessing resources* of a term M is the set of type-variables which occur in in the left component Y of any sub-term $\langle Y, \phi, N\rangle_*$. This can be made precise with a recursive definition.

The rules for existentials, which follow the generalized forms for natural deduction introduced by Prawitz [12], are presented in Figure 3. Both of $(\exists E)$ and $(\exists_* E)$ are subject to the side-condition $\alpha \notin FTV(\Delta) \cup FTV(\sigma)$, which is standard for the elimination of existentials. In addition, both are subject to the side-condition $WR(M) \cup WR(N) = \emptyset$, because of the presence of the multiplicative. Furthermore, the condition $\alpha \notin WR(N)$ is required for $(\exists_* E)$.

$$(T\exists) \quad \frac{X; \alpha \vdash \tau}{X \vdash \exists\alpha.\tau} \qquad \frac{X \mid \Gamma \vdash M : \exists\alpha.\tau \quad X; \alpha \mid \Delta(x:\tau) \vdash N : \sigma}{X \mid \Delta(\Gamma) \vdash \mathbf{unpack}\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ N : \sigma} \quad (\exists E)$$

$$(\exists I) \quad \frac{X \mid \Gamma \vdash (M:\tau)[\phi/\alpha] \qquad X \vdash \exists\alpha.\tau}{X \mid \Gamma \vdash \langle\phi, M\rangle : \exists\alpha.\tau}$$

$$(T\exists_*) \quad \frac{X, \alpha \vdash \tau}{X \vdash \exists_*\alpha.\tau} \qquad \frac{X \mid \Gamma \vdash M : \exists_*\alpha.\tau \quad X, \alpha \mid \Delta(x:\tau) \vdash N : \sigma}{X \mid \Delta(\Gamma) \vdash \mathbf{unpack}_*\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ N : \sigma} \quad (\exists_* E)$$

$$(\exists_* I) \quad \frac{X, Y(Z) \mid \Gamma \vdash (M:\tau)[\phi/\alpha] \quad Y(Z) \vdash \phi \quad X, Z \vdash \Gamma \quad X \vdash \exists_*\alpha.\tau}{X, Z \mid \Gamma \vdash \langle Y(Z), \phi, M\rangle_* : \exists_*\alpha.\tau}$$

Fig. 3. Existential rules

The additive quantifier behaves essentially as the standard polymorphic existential. The multiplicative is more unusual. This partially hides the resources (type variables) used in its formation. The work on first-order **BI** suggests a form in which Y is completely hidden. This rule is derivable from the one given. The more general version is adopted in order to give a corresponding η-rule.

The $\beta\eta$-conversions for existentials are

$$\begin{aligned}
(X \mid \mathbf{unpack}\ \langle\phi, M\rangle\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ N) &\to_\beta (X \mid N[M/x][\phi/\alpha]) \\
(X \mid \mathbf{unpack}\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ (N[\langle\alpha, x\rangle/z])) &\to_\eta (X \mid N[M/z]) \\
(X \mid \mathbf{unpack}_*\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ (N[\langle\alpha, \alpha, x\rangle_*/z]) &\to_\eta (X \mid N[M/z]) \\
(X, Z \mid \mathbf{unpack}_*\ \langle Y(Z), \phi, M\rangle_*\ \mathbf{as}\ \langle\alpha, x\rangle\ \mathbf{in}\ N) &\to_\beta (X, Y(Z) \mid N[M/x][\phi/\alpha])
\end{aligned}$$

and suitable ζ-conversions for existentials are also possible, provided no universal quantifiers are present. Notice how the fibre changes in the β-conversion for the multiplicative. Let $\twoheadrightarrow$ be the reduction relation generated by $\to_\beta$ and $\to_\eta$.

Most of the metatheory goes through as it did for the system with universals rather than existentials. In particular, strong normalization can again be proved

by the translation method. However, there are a few important changes, notably to substitution and subject-reduction.

Proposition 5. *If* $X \mid \Gamma(x:\tau) \vdash N : \sigma$ *and* $X \mid \Delta \vdash M : \tau$ *are both derivable and* $WR(M) \cap WR(N) = \emptyset$ *then* $X \mid \Gamma(\Delta) \vdash N[M/x] : \sigma$ *is derivable.*

The condition on the substitution law forces us to place the side-condition $WR(M) \cap WR(N) = \emptyset$ on the binary elimination rules $(\wedge E)$, $(\rightarrow E)$, $(*E)$, $(-\!\!*E)$ and is the reason why we need the same condition for the existentials.

Proposition 6. *If* $X \mid \Gamma \vdash M : \tau$ *is derivable and* $(X \mid M) \twoheadrightarrow (Y \mid N)$ *then* $Y \mid \Gamma \vdash N : \tau$ *is derivable.*

The existential does not have a simple $\alpha 2\lambda 2$-hyperdoctrine interpretation and, in particular, we cannot just use a left-adjoint to the pseudoprojection substitution. However, an interpretation can be given to each judgement by requiring the existence of certain assignments and arrows.

The introduction rule $(\exists_* I)$ for the multiplicative existential hides not only the representation type, but also the resources associated with the representation type. Once hidden, these resources are not visible to terms formed over the same fibre (see the substitution rule) and are only revealed by a subsequent use of the elimination rule $(\exists_* E)$, leading to a fibre-changing β-conversion, as above. In this respect the formation of multiplicative existentials is reminiscent of the formation of function closures. Furthermore, the elimination of $\exists_*$ is reminiscent of the application of function closures, though perhaps with some side-effects.

We conjecture that bunched polymorphism is an appropriate setting to develop type systems for memory-management. One approach to this is alias typing [18] which allows the programmer to issue instructions that safely allocate and deallocate chunks of memory, known as locations. Locations are used as parameters in types, for example $x : ptr(l)$, which asserts that a program variable is a pointer to the location l. A form of polymorphism is introduced through the use of location variables, which range over locations. Instructions are typed in contexts of *aliasing constraints*: these specifiy the types of entities contained in certain locations and location variables. It is difficult to formalize direct translations of such systems into the bunched setting because of the complexity of their type systems. However, it seems relatively clear that what the authors intend to enforce are non-sharing (anti-aliasing) constraints on chunks of memory. Consider, for example, the following statement, taken from [18].

> The existential $\exists[\rho : Loc \mid \{\rho \mapsto \tau_1\}].\tau_2$ may be read "there exists some location ρ, different from all others in the program, such that ρ contains an object of type τ_1, and the value contained in this data structure has type τ_2."

What is intended to be different is surely not the location variable ρ itself, but rather the memory assigned to it by the environment. Under such a reading, it would seem more appropriate to use bunching rather than linearity as a foundation for the type system. Bunched alternatives to the linear approaches to type systems for *regions* [17], [19], should be equally interesting.

References

1. B.J. Day. On closed categories of functors. In *Lecture Notes in Mathematics* 137, pages 1–38. Springer-Verlag, Berlin-New York, 1970.
2. B.J. Day. An embedding theorem for closed categories. In *Lecture Notes in Mathematics* 420, pages 55–65. Springer-Verlag, Berlin, 1973.
3. J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proc. 2nd Scandinavian Logic Symposium*, pages 63–92. North-Holland, 1971.
4. J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.
5. J.-Y. Girard. Linear logic. *Theoretical Computer Science* 50, pages 1–102, 1987.
6. M.J. Gabbay and A.M.Pitts A new approach to abstract syntax and variable binding. *Formal Aspects of Computing* 13, pages 341–363, 2002.
7. J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40:135–165, 1988.
8. B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.
9. J.C. Mitchell and G.D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10:470–502, 1988.
10. P. O'Hearn. On bunched typing. *J. Functional Programming.*, 13:747–796, 2003.
11. P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
12. D. Prawitz. Proofs and the meaning and completeness of logical constants. In *Essays on mathematical and philosophical logic*, pages 25–40. D. Reidel, 1978.
13. D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002. Errata at: http://www.cs.bath.ac.uk/∼pym/BI-monograph-errata.pdf.
14. J.C. Reynolds. Towards a theory of type structure. In *Lecture Notes in Computer Science* 19, pages 408–425. Springer, 1974.
15. J.C. Reynolds. Separation logic: a logic for shared mutable data structure. In *Proc. LICS '02*, pages 55–74. IEEE Computer Science Press, 2002.
16. R.A.G. Seely. Categorical semantics for higher order polymorphic lambda calculus. *Journal of Symbolic Logic*, 52:969–989, 1987.
17. M. Tofte and J.-P. Talpin Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.
18. D. Walker and J.G. Morrisett. Alias types for recursive data structures. In *Lecture Notes in Computer Science* 2071, pages 177–206. Springer-Verlag, 2001.
19. D. Walker and K. Watkins On regions and linear types In *Proc. International Conference on Functional Programming*, 181-192. 2001.