Under consideration for publication in Math. Struct. in Comp. Science

Bunched Polymorphism

MATTHEW COLLINSON[1] , DAVID PYM[1] , EDMUND ROBINSON[2]

[1] *Hewlett-Packard Laboratories, Bristol, BS34 8QZ, UK.*

[2] *Queen Mary, University of London, E1 4NS, UK.*

Received 17th August 2007; Revised 12th May 2008

We describe a polymorphic, typed lambda calculus with substructural features. This calculus extends the first-order substructural lambda calculus $\alpha\lambda$ associated with bunched logic. A particular novelty of our new calculus is the substructural treatment of second-order variables. This is accomplished through the use of bunches of type variables in typing contexts. Both additive and multiplicative forms of polymorphic abstraction are then supported. The calculus has sensible proof-theoretic properties and a straightforward categorical semantics using indexed categories. We produce a model for additive polymorphism with first-order bunching based on partial equivalence relations. We consider additive and multiplicative existential quantifiers separately from the universal quantifiers.

1. Introduction

Formal languages with substructural inference mechanisms are powerful tools in Logic and Computer Science. They can be characterized by the absence or restriction of the traditional structural proof rules of weakening, contraction and exchange. Perhaps the best known substructural logic is Linear Logic (Girard 1987). This has a strong proof-theoretic justification, but also admits a *resource reading*, which concerns the number of uses of the resources (assumed formulae) in deriving a formula. This interpretation gives rise to many applications.

Bunched logic (O'Hearn and Pym 1999; Pym 2002; Pym *et al.* 2004), abbreviated **BI**, is another substructural logic. It also has a resource reading, called the *sharing interpretation.* Here the resource reading concerns sharing relationships on the resources used by a formula. This is achieved by decomposing logical connectives into *additive/multiplicative* pairs: for example conjunction, $\wedge$, becomes $(\wedge, *)$, and implication, $\rightarrow$, becomes $(\rightarrow, -\!\!*)$. The additive variants are intended to be like their traditional, structural versions. In contrast, the multiplicatives are subject to certain sharing constraints: for example, in a formula $\phi * \psi$, the two formulae ϕ and ψ must not share resources. This logic is the foundation of the assertion language in Separation Logic (Reynolds 2002) which gives an efficient Floyd-Hoare logic for imperative programs with mutable state.

Bunched logic has a good proof-theory and a natural notion of categorical model. These models can be realized in certain functor categories based on sets, and this gives rise to a host of applications. Some of these functor categories are closely related to possible-worlds models for higher-order languages with store (Reynolds 1981; Oles 1982). The bunched type theory $\alpha\lambda$ corresponding to **BI** has been developed in (O'Hearn 2003; Pym 2002). In conjunction with the sharing interpretation and possible-world models it has been used in (O'Hearn 2003) to give a refined treatment of Idealized Algol, **IA**, and a treatment of memory operations on references in functional languages (Berdine and O'Hearn 2006).

Polymorphic quantifiers and *polymorphic lambda-calculi* were introduced in (Girard 1971; Girard 1972) and again in (Reynolds 1974), where their use in programming was emphasized. In logics of this kind formulae have second-order propositional variables, that is, variables that may be instantiated with propositions. Polymorphic lambda-calculi and programs may contain second-order variables (also known as *type variables*) that may be instantiated with types. Such lambda-calculi also have a mechanism for *polymorphic abstraction*: this is used to construct functions with type variables as arguments. These calculi have a semantics in certain indexed categories. A particular instance of such a model is constructed using *partial equivalence relations* (PERs) on the natural numbers, see for example the early work by Girard (Girard 1972) or the modern treatment in Jacobs book (Jacobs 1999).

In this paper we study combinations of bunching and polymorphism. We not only combine ordinary polymorphism with $\alpha\lambda$, but also make a bunched treatment of the second-order variables. Bunching at the first and second-order levels are essentially orthogonal, so there are perfectly sensible calculi with bunching at both, either or neither of the two levels. The presence of bunching at second-order means that there are additive and multiplicative variants of polymorphism within the calculus. There are additive polymorphic types $\forall\alpha.\tau$, polymorphic abstraction terms $\Lambda\alpha.M : \forall\alpha.\tau$ and instantiations $\mathrm{App}(M, \sigma) : \tau[\sigma/\alpha]$ that behave in essentially the same way as ordinary polymorphism. There are also multiplicative polymorphic types $\forall_*\alpha.\tau$, terms $\Lambda_*\alpha.M : \forall_*\alpha.\tau$ and instantiations of the form $\mathrm{App}_*(M, A, \sigma) : \tau[\sigma/\alpha]$. We give a detailed account of the theory and models of such calculi by extending carefully the standard treatments for ordinary polymorphism. We also study polymorphic existential quantifiers as these are well-known to be related to abstract data-types (Mitchell and Plotkin 1988). An additive existential, $\exists$, that mimics ordinary existential types may be added as may a multiplicative variant, $\exists_*$. However, the form of the proof rules for $\exists_*$ suggested by the proof-theory does not agree precisely with that which would fit naturally with the indexed category approach.

In §2 we introduce the most basic bunched polymorphic lambda calculus, $\alpha 2\lambda 2$, its derivation rules and reductions. In §3 we study the metatheory associated with this calculus. We provide appropriate versions of basic proof-theoretic results such as substitution (cut-elimination), subject-reduction and normalization. In §4 we present categorical models for $\alpha 2\lambda 2$ in the form of hyperdoctrines (indexed categories). We prove soundness and completeness theorems for the calculus in such models. In §5 we study polymorphism in a category of partial equivalence relations. We show that additive polymorphism with first-order bunching may be interpreted in this category. A calculus with bunched polymorphic existential quantifiers is studied in §6. The paper concludes with a summary and a discussion of further avenues and open problems in §7.

The work reported herein are results from the project 'Bunched ML', funded by the United Kingdom EPSRC and carried out at the University of Bath and Queen Mary, University of London. We wish to acknowledge suggestions given by our collaborators Josh Berdine and Peter O'Hearn and support given by HP Labs, Bristol. We also wish to thank the anonymous referees for producing a number of helpful comments.

2. The Calculus

The main bunched polymorphic lambda calculus, which we shall call $\alpha 2\lambda 2$, is an extension of the bunched, simply-typed calculus $\alpha\lambda$ presented in (O'Hearn 2003) and (Pym 2002). Bunches are contexts that are structured in a particular way. The calculus $\alpha 2\lambda 2$ produces judgements of the form

$$X \mid \Gamma \vdash M : \tau$$

saying that a term M is typed with τ, given a bunch of type variables X and a bunch of (ordinary) variables Γ.

Let us first consider bunches in general. Suppose that we have some given syntactic category $\mathcal{B}$. A *bunch* of $\mathcal{B}$-elements is a binary tree with each leaf labelled by a $\mathcal{B}$-element or one of the symbols $\emptyset$ or $\emptyset_*$, and each internal node labelled with either of the symbols ';' or ','. Thus bunches are generated by the grammar

$$B := \emptyset \mid \emptyset_* \mid b \mid B; B \mid B, B$$

where b ranges over $\mathcal{B}$-elements. The symbols ';' and ',' are referred to as *additive* and *multiplicative combination*, respectively. The bunches $\emptyset$ and $\emptyset_*$ are, respectively, the *additive* and *multiplicative units*. The formation of bunches is sometimes subject to the restriction that any $\mathcal{B}$-element may occur at most once in any bunch. We refer to this as the *linearity condition.*

A *sub-bunch* B' of a bunch B is a subtree of B such that all leaves of B' are leaves of B. Let $B(B_1 \mid \ldots \mid B_n)$ be the notation for a bunch B with distinguished, non-overlapping sub-bunches $B_1, \ldots, B_n$. We write $B[B'_1/B_1, \ldots B'_n/B_n]$, or sometimes, $B(B'_1 \mid \ldots \mid B'_n)$ for the bunch formed by replacing each bunch B_i in B with a bunch B'_i. For any bunch $B(B_1 \mid B_2)$ the bunches B_1 and B_2 are combined at some minimal internal node. If this node is labelled with ';' then we say that B_1 and B_2 are *additively combined* in B or write $B_1; B_2$ in B for short. If it is labelled with ',' then we say that they are *multiplicatively combined* in B or write B_1, B_2 in B. We use these notations sparingly since, for example, B_1, B_2 in B does not entail that B_1, B_2 is a sub-bunch of B.

A pair of equivalence relations normally appear with bunches. The first equivalence, $\equiv$, is used to build structural rules that allow us to permute leavs. It is a congruence generated by commutative monoid rules for ';' and ','. Thus the commutative monoid axioms can be applied at arbitrary depth in any bunch. To be precise $\equiv$ is generated by the rules

— if $B_2 \equiv B_3$ then $B[B_3/B_1] \equiv B[B_2/B_1]$, where $B(B_1)$ is any bunch
— if $B_1 \equiv B_2$ and $B_2 \equiv B_3$ then $B_1 \equiv B_3$
— $B \equiv B; \emptyset$ and $B \equiv B, \emptyset_*$

— $B_1; (B_2; B_3) \equiv (B_1; B_2); B_3$ and $B_1, (B_2, B_3) \equiv (B_1, B_2), B_3$
— $B_1; B_2 \equiv B_2; B_1$ and $B_1, B_2 \equiv B_2, B_1$

on bunches. Additional axioms may be required in specific situations. For example, we often wish to consider *affine* bunches, in which $\emptyset \equiv \emptyset_*$ is required.

The second equivalence relation, $\cong$, is used to control contraction in the derivation of judgements. We discuss the relations $\cong$ used with bunches of type variables and variables below separately, since they are slightly different.

Assume a countable collection of *type variables* to be given. We allow the letters α, β to range over type variables. A *hub* is an affine bunch of type variables, subject to the linearity condition. For the sake of brevity we use the unit $\emptyset$ but not $\emptyset_*$, since hubs are affine. Thus hubs are generated as follows

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

such that every type variable may occur at most once in a bunch. Let the letters X, Y, Z range over hubs.

The equivalence $\cong$ on hubs is simply renaming of type variables: $X \cong Y$ if Y can be obtained from X by renaming bijectively with type variables. Notice that both X and Y then have the same internal structure.

The types of the calculus are generated by

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

where α is a type variable. The constructors $\top$, $\wedge$, $\to$ and $\forall$ are referred to, respectively, as the *additive unit*, *product*, *abstraction* and *polymorphic abstraction* (*universal quantifier*). Matching these, there are *multiplicative unit* I, *product* $*$, *abstraction* $\mathrel{-\!\!*}$ and *polymorphic abstraction* (*universal quantifier*) $\forall_*$. The additive *sum*, $\vee$, is also included. Let the letters σ, τ, υ range over types.

Assume a countable collection of *variables* to be given. Let x, y, z range over these variables. A *(typing) context* is a bunch of typed variables, subject to a linearity condition. Contexts are generated by the grammar

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

where x is a variable, τ is a type and any variable occurs at most once. On the other-hand, types may occur more than once in a bunch — for example, $x : \tau, y : \tau$ is a bunch for any type τ. Note that contexts are *not* required to be affine, in general, but that requiring this also leads to a sensible, specialized calculus. This is similar to the situation for $\alpha\lambda$. The units $\emptyset$ and $\emptyset_*$ are distinct from each other, and from the unit $\emptyset$ for hubs. These contexts are nothing more than the contexts of $\alpha\lambda$, but such that types may contain type variables and be formed using the polymorphic quantifiers. The letters Γ and Δ are reserved for contexts.

The relation $\Gamma \cong \Delta$ between contexts holds just when Δ can be obtained by a type-preserving, bijective relabelling of the leaves of Γ. That is, any leaf $x : \tau$ of Γ must correspond to a unique leaf $y : \tau$ of Δ.

The terms of the language are given by the following grammar

$$\begin{aligned} M := \;& x \mid \top \mid I \mid \text{ let } I \text{ be } M \text{ in } M \\ & \mid \text{inl}(M) \mid \text{inr}(M) \mid \text{case } M \text{ of inl}(x) \Rightarrow M \text{ or inr}(y) \Rightarrow M \\ & \mid \langle M, M\rangle \mid \pi_1 M \mid \pi_2 M \mid M * M \mid \text{ let } (x,y) \text{ be } M \text{ in } M \\ & \mid \lambda x : \tau.M \mid \text{app}(M,M) \mid \lambda_* x : \tau.M \mid \text{app}_*(M,M) \\ & \mid \Lambda\alpha.M \mid \text{App}(M,\tau) \mid \Lambda_*\alpha.M \mid \text{App}_*(M,A,\tau) \;\;, \end{aligned}$$

where τ is a type, x is a variable, α is a type variable and A is a finite set of type-variables. The forms which we have added to $\alpha\lambda$ are the *additive polymorphic abstraction* $\Lambda\alpha.M$, *multiplicative polymorphic abstraction* $\Lambda_*\alpha.M$, *additive instantiation* $\text{App}(M,\tau)$, and *multiplicative instantiation* $\text{App}_*(M,A,\tau)$.

Let $\mathit{fv}(-)$ be the operation that produces the set of variables that occur in a context or free in a term. We use the notation $\mathit{ftv}(-)$ for the operation that returns the set of type variables which occur in a hub, or free in a type, or free in at least one of the types of the variables in a context. In particular, in the types $\forall\alpha.\tau$, $\forall_*\alpha.\tau$ the type variable α is not free and

$$\mathit{ftv}(\forall_*\alpha.\tau) = \mathit{ftv}(\forall\alpha.\tau) = \mathit{ftv}(\tau) \smallsetminus \{\alpha\}$$

holds.

We define the set of free type variables $\mathit{ftv}_\Gamma(M)$ that are free in a term as follows given in a context Γ that includes all of the variables that are free in M. The idea is that the set $\mathit{ftv}_\Gamma(M)$ will include all free types of sub-terms used in the construction of M, not just those types appearing in the type of the term. The recursive definition is as follows:

$$\begin{aligned} &\mathit{ftv}_\Gamma(\top) = \mathit{ftv}_\Gamma(I) = \emptyset \\ &\mathit{ftv}_\Gamma(x) = \mathit{ftv}(\tau) \text{ where } x : \tau \text{ is in } \Gamma \\ &\mathit{ftv}_\Gamma(\text{ let } I \text{ be } M \text{ in } N) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N) \\ &\mathit{ftv}_\Gamma(\text{inl}(M)) = \mathit{ftv}_\Gamma(\text{inr}(M)) = \mathit{ftv}_\Gamma(M) \\ &\mathit{ftv}_\Gamma(\text{case } M \text{ of inl}(x) \Rightarrow N_1 \text{ or inr}(y) \Rightarrow N_2) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N_1) \cup \mathit{ftv}_\Gamma(N_2) \\ &\mathit{ftv}_\Gamma(\langle M, N\rangle) = \mathit{ftv}_\Gamma(M * N) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N) \\ &\mathit{ftv}_\Gamma(\pi_1 M) = \mathit{ftv}_\Gamma(\pi_2 M) = \mathit{ftv}_\Gamma(M) \\ &\mathit{ftv}_\Gamma(\text{ let } (x,y) \text{ be } M \text{ in } N) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N) \\ &\mathit{ftv}_\Gamma(\lambda x : \sigma.M) = \mathit{ftv}_\Gamma(\lambda_* x : \sigma.M) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}(\sigma) \\ &\mathit{ftv}_\Gamma(\text{app}(M,N)) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N) \\ &\mathit{ftv}_\Gamma(\text{app}_*(M,N)) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}_\Gamma(N) \\ &\mathit{ftv}_\Gamma(\Lambda\alpha.M)) = \mathit{ftv}_\Gamma(\Lambda_*\alpha.M)) = \mathit{ftv}_\Gamma(M) \smallsetminus \alpha \\ &\mathit{ftv}_\Gamma(\text{App}(M,\sigma)) = \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}(\sigma) \\ &\mathit{ftv}_\Gamma(\text{App}_*(M,A,\tau)) = A \cup \mathit{ftv}_\Gamma(M) \cup \mathit{ftv}(\tau) \;. \end{aligned}$$

We usually drop the subscript Γ and write $\mathit{ftv}(M)$ when the context Γ is clear.

We write $X \vdash \tau$ and say that τ *is well-formed over* X whenever $\mathit{ftv}(\tau) \subseteq \mathit{ftv}(X)$. Similarly, write $X \vdash \Gamma$ and say Γ *is well-formed over* X whenever $\mathit{ftv}(\Gamma) \subseteq \mathit{ftv}(X)$ holds.

We introduce a syntactic measure μ which assigns to each term the set of type variables that are free and occur in some application of the multiplicative universal quantifier. This may be defined recursively as follows:

$$\begin{aligned}
&\mu(\top) = \mu(I) = \mu(x) = \emptyset\\
&\mu(\langle M, N\rangle) = \mu(M * N) = \mu(M) \cup \mu(N)\\
&\mu(\text{ let } (x, y) \text{ be } M \text{ in } N) = \mu(M) \cup \mu(N)\\
&\mu(\text{inl}(M)) = \mu(\text{inr}(M)) = \mu(M)\\
&\mu(\text{case } M \text{ of } \text{inl}(x) \Rightarrow N_1 \text{ or } \text{inr}(y) \Rightarrow N_2) = \mu(M) \cup \mu(N_1) \cup \mu(N_2)\\
&\mu(\text{ let } I \text{ be } M \text{ in } N) = \mu(M) \cup \mu(N)\\
&\mu(\text{app}(M, N)) = \mu(M) \cup \mu(N)\\
&\mu(\text{app}_*(M, N)) = \mu(M) \cup \mu(N)\\
&\mu(\pi_1 M) = \mu(\pi_2 M) = \mu(M)\\
&\mu(\lambda x : \sigma.M) = \mu(\lambda_* x : \sigma.M) = \mu(M)\\
&\mu(\Lambda\alpha.M) = \mu(\Lambda_*\alpha.M) = \mu(M) \setminus \{\alpha\}\\
&\mu(\text{App}(M, \tau)) = \mu(M)\\
&\mu(\text{App}_*(M, A, \tau)) = \mu(M) \cup A.
\end{aligned}$$

We call $\mu(M)$ the *instantiation set* of M.

The typing of terms uses the type and context formation judgements. The *term formation judgements* are derived from the system of rules shown in Figure 1. We write $X \mid \Gamma \vdash M : \tau$ and say that M *is well-formed with type* τ *over* X *and* Γ if it is produced by this system.

The rules ($\top I$), (II) introduce the additive and multiplicative units. Together with (Ax) they are the rules that occur at the leaves of derivations. The rules ($\top E$) and (IE) eliminate the units. The rules ($\wedge I$) and ($\wedge E$) give the additive product while the rules ($*I$) and ($*E$) give the multiplicative product. The rules for additive and multiplicative functions are ($\to I$), ($\to E$), ($-\!*I$) and ($-\!*E$). The additive coproduct is given by the rules ($\vee I$) and ($\vee E$). There are structural rules of exchange (E), weakening (W), and contraction (C) for bunches of (ordinary) variables. There are rules for additive polymorphism ($\forall I$), ($\forall E$) and multiplicative polymorphism ($\forall_* I$), ($\forall_* E$). Finally there are structural rules of exchange ($E2$), weakening ($W2$) and contraction ($C2$) for hubs.

Notice that all of the first-order rules have been presented relative to a fixed hub X. They are essentially the familiar rules for $\alpha\lambda$, but parametrised by the hub.

Many of the rules are subject to side-conditions. These will sometimes be written as additional premises of rules. The side-condition on the rules (Ax), ($\forall E$) and ($\forall_* E$) ensure that in a derivation $X \mid \Gamma \vdash M : \tau$ all of the free type variables of τ occur in X. Together with the side-conditions on (W), ($\forall I$) and ($\forall_* I$) we have that all of the free type variables in Γ occur in X. The elimination rules ($\top E$), (IE), ($\wedge E$), ($*E$), ($\to E$), ($-\!*E$) are each subject to a side-condition

$$\mu(N) \cap \mathit{ftv}(M) = \emptyset \qquad (\dagger)$$

that requires the separation of certain free type variables. In a similar way, the rule ($\vee E$) requires the conditions ($\dagger_i$), being $\mu(N_i) \cap \mathit{ftv}(M) = \emptyset$, for both $i = 1, 2$. These eliminations work through substitutions and, when substituting, we must ensure that the disjointness conditions on type variables in N are not violated. Such side-conditions are necessary in order to have a standard substitution property for terms, namely that a

$$(\top I)\quad \frac{}{X \mid \emptyset \vdash \top : \top} \qquad (Ax)\quad \frac{}{X \mid x : \tau \vdash x : \tau}\ (X \vdash \tau) \qquad \frac{}{X \mid \emptyset_* \vdash I : I}\quad (II)$$

$$(TE)\quad \frac{X \mid \Gamma(\emptyset) \vdash N : \tau \quad X \mid \Delta \vdash M : \top}{X \mid \Gamma(\Delta) \vdash N[M/\top] : \tau} \quad (\dagger) \quad \frac{X \mid \Gamma(\emptyset_*) \vdash N : \tau \quad X \mid \Delta \vdash M : I}{X \mid \Gamma(\Delta) \vdash \text{let } I \text{ be } M \text{ in } N : \tau}\quad (IE)$$

$$(\wedge I)\quad \frac{X \mid \Gamma \vdash M : \sigma \quad X \mid \Delta \vdash N : \tau}{X \mid \Gamma; \Delta \vdash \langle M, N\rangle : \sigma \wedge \tau} \qquad \frac{X \mid \Gamma \vdash M : \sigma \quad X \mid \Delta \vdash N : \tau}{X \mid \Gamma, \Delta \vdash M * N : \sigma * \tau}\quad (*I)$$

$$(\wedge E)\quad \frac{X \mid \Gamma; (x : \phi; y : \psi) \vdash N : \tau \quad X \mid \Gamma \vdash M : \phi \wedge \psi}{X \mid \Gamma \vdash N[\pi_1 M/x, \pi_2 M/y] : \tau}\quad (\dagger)$$

$$(*E)\quad \frac{X \mid \Gamma(x : \phi, y : \psi) \vdash N : \tau \quad X \mid \Delta \vdash M : \phi * \psi}{X \mid \Gamma(\Delta) \vdash \text{let } (x, y) \text{ be } M \text{ in } N : \tau}\quad (\dagger)$$

$$(\rightarrow I)\quad \frac{X \mid \Gamma; x : \sigma \vdash M : \tau}{X \mid \Gamma \vdash \lambda x : \sigma . M : \sigma \rightarrow \tau} \qquad \frac{X \mid \Gamma, x : \sigma \vdash M : \tau}{X \mid \Gamma \vdash \lambda_* x : \sigma . M : \sigma \mathrel{-\!\!*} \tau}\quad (\mathrel{-\!\!*} I)$$

$$(\vee Il)\quad \frac{X \mid \Gamma \vdash M : \sigma}{X \mid \Gamma \vdash \text{inl}(M) : \sigma \vee \tau} \qquad \frac{X \mid \Gamma \vdash M : \tau}{X \mid \Gamma \vdash \text{inr}(M) : \sigma \vee \tau}\quad (\vee Ir)$$

$$(\vee E)\quad \frac{X \mid \Gamma \vdash M : \tau_1 \vee \tau_2 \quad X \mid \Delta(x : \tau_1) \vdash N_1 : \sigma \quad X \mid \Delta(y : \tau_2) \vdash N_2 : \sigma}{X \mid \Delta(\Gamma) \vdash \text{case } M \text{ of inl}(x) \Rightarrow N_1 \text{ or inr}(y) \Rightarrow N_2 : \sigma}\quad (\dagger_1, \dagger_2)$$

$$(\rightarrow E)\quad \frac{X \mid \Gamma \vdash N : \sigma \rightarrow \tau \quad X \mid \Delta \vdash M : \sigma}{X \mid \Gamma; \Delta \vdash \text{app}(N, M) : \tau} \quad (\dagger) \quad \frac{X \mid \Gamma \vdash N : \sigma \mathrel{-\!\!*} \tau \quad X \mid \Delta \vdash M : \sigma}{X \mid \Gamma, \Delta \vdash \text{app}_*(N, M) : \tau}\quad (\mathrel{-\!\!*} E)$$

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

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

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

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

$$(C)\quad \frac{X \mid \Gamma(\Delta; \Delta') \vdash M : \tau}{X \mid \Gamma(\Delta) \vdash M[\Delta/\Delta'] : \tau}\quad (\Delta \cong \Delta') \qquad (X \equiv Y)\quad \frac{X \mid \Gamma \vdash M : \tau}{Y \mid \Gamma \vdash M : \tau}\quad (E2)$$

$$(C2)\quad \frac{X(Y; Y') \mid \Gamma \vdash M : \tau}{X(Y) \mid \Gamma[Y/Y'] \vdash M[Y/Y'] : \tau[Y/Y']}\quad (Y \cong Y') \qquad \frac{Y \mid \Gamma \vdash M : \tau}{X(Y) \mid \Gamma \vdash M : \tau}\quad (W2)$$

Fig. 1. Term formation rules

substitution of a well-formed term for a first-order variable in another well-formed term produces a well-formed term. The asymmetry of this side-condition (between M and N) comes from the asymmetry of substitution (of, say, M, for some variable in N). This is explained in more detail below.

It should be noted that the calculus only allows weakening (W) around ';' in first-order contexts. In contrast rule $(W2)$ is equivalent to having weakening around both ';' and ',' in a hub. We say that the calculus is *affine* in the hubs. This is why we use just one unit for hubs — more discussion of affine calculi may be found in (O'Hearn and Pym 1999; O'Hearn 2003; Pym 2002). It appears that hubs are required to be affine in order to get a sensible proof theory. In particular, it seems necessary for the substitution property (admissibility of the cut-rule). This property is given for $\alpha 2\lambda 2$ in Proposition 1 below.

Contraction is not available around a multiplicative combination. As a consequence we may regard a hub X, Y as consisting of two hubs whose variables are strongly distinguished. The multiplicative quantifier differs from the additive quantifier in that we may only instantiate with types whose variables are thus distinguished. This lifts the sharing interpretation for bunched calculi (O'Hearn and Pym 1999) to the second-order level.

Let X be a bunch and A be a set of type variables. Let $X{\restriction}_A$ be the bunch formed by:

— erasing all leaves of X that feature a variable not in A
— deleting edges into such leaves
— if an internal node is not branching then deleting that node and replacing the two incident edges with a single edge between the two previously adjacent vertices.
— if everything is deleted then insert a $\emptyset$.

Introduce the notation $X \lhd Y$ to mean that Y can be recovered from X by weakenings and equivalences. To be precise Y can be produced from X by attaching a finite number of bunches using the constructors ';' and ',', and using the congruence rules for $\equiv$.

Recall that bunches are defined here as (binary) trees, and a sub-bunch X of a bunch Y is defined to be a subtree X of the tree Y. Note that if X is a sub-bunch of Y then $X \lhd Y$, but not vice-versa.

Definition 1. Let X and Y be bunches and $\mathit{ftv}(Y) = \{\beta_1, \ldots, \beta_n\}$. A *(type) substitution* $s : X \longrightarrow Y$ is a sequence

$$[\sigma_1/\beta_1, \ldots, \sigma_n, \beta_n]$$

such that

— $X \vdash \sigma_j$ for each $1 \leq j \leq n$
— if Y_1 and Y_2 are bunches such that $Y_1, Y_2 \lhd Y$, then there are bunches X_1 and X_2 with $X_1, X_2 \lhd X$ and, for $i = 1, 2$ and $1 \leq j \leq n$, if $\beta_j \in \mathit{ftv}(Y_i)$ then $X_i \vdash \sigma_j$.

We sometimes use the usual liberal variant of the above sequence notation for substitution in which only the variables which are not replaced by themselves are mentioned.

It follows that substitutions have the following property: if β_i and β_j are multiplicatively combined in Y then $\mathit{ftv}(\sigma_i) \cap \mathit{ftv}(\sigma_j) = \emptyset$. This reinforces the sharing interpretation for type variables in such calculi, as the multiplicative combination of type variables requires that they use disjoint resources.

Any substitution s as above defines a map s^* from sets of type variables in Y to sets of type variables in X. This is given by

$$s^*(A) = \bigcup_{i \in \{i \mid \beta_i \in A\}} \mathit{ftv}(\sigma_i)$$

for any A that is a finite subset of Y.

The substitution $s : X \longrightarrow Y$ maps any type $Y \vdash \tau$ to a type $X \vdash s^*(\tau)$ by simultaneously replacing each β_j with σ_j for $1 \leq j \leq n$. We also write $\tau[\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n]$ for $s^*(\tau)$. This assignment is defined to work in the usual way. For example, we have

$$\begin{array}{rcl} s^*(\sigma \mathrel{-\!\!*} \tau) & = & s^*(\sigma) \mathrel{-\!\!*} s^*(\tau) \\ \\ s^*(\forall_* \alpha.\tau) & = & \forall_* \alpha.s^*(\tau) \end{array}$$

for multiplicative function and polymorphic types.

We extend the substitution to act on contexts: any context $Y \vdash \Gamma$ is mapped to a context $X \vdash s^*(\Gamma)$ where $s^*(\Gamma)$ is formed by re-typing the variables of Γ with the images of the original types under s^*. That is, $x : \tau$ is in Γ if and only if $x : s^*(\tau)$ is at the corresponding node in $s^*(\Gamma)$ and all internal nodes (';' and ',') and units are unchanged.

The substitution $s : X \longrightarrow Y$ maps any term M with free type variables in Y to one with free type variables in X. The substitution is defined in the usual way: in particular, we have

$$\begin{array}{rcl} s^*(\Lambda_* \alpha.M) & = & \Lambda_* \alpha.s^*(M) \\ \\ s^*(\mathrm{App}_*(M, A, \tau)) & = & \mathrm{App}_*(s^*(M), s^*(A), s^*(\tau)) \end{array}$$

for dealing with the multiplicative polymorphic abstraction. An important consequence of this definition is that substitution commutes with both kinds of polymorphic abstraction. We shall see later that under such a substitution any well-formed term $Y \mid \Gamma \vdash M : \tau$ is mapped to a well-formed term $X \mid s^*(\Gamma) \vdash s^*(M) : s^*(\tau)$. Notice that in the rule $(\forall_* E)$ the context of the conclusion is extended with the entire hub of the type with which we are instantiating, but that the substitution used to perform the instantiation only uses the free type variables of the instantiating type.

For any hub Y, with say $\mathit{ftv}(Y) = \{\beta_1, \ldots, \beta_n\}$ the sequence $[\beta_1/\beta_1, \ldots, \beta_n/\beta_n]$ is the *identity* substitution. For any pair of substitutions

$$s = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y \qquad t = [\tau_1/\alpha_1, \ldots, \tau_p/\alpha_p] : Y \longrightarrow Z$$

there is a composite substitution

$$t \circ s = [s^*(\tau_1)/\alpha_1, \ldots, s^*(\tau_p)/\alpha_p] : X \longrightarrow Z$$

from types over Z to types over X. Furthermore, the lemma below holds.

Lemma 1. There is a category with hubs as objects and type substitutions as arrows.

If $X \cong Y$ then there is a substitution $s : Y \longrightarrow X$, written $[Y/X]$, given by replacing every free type variable from X with the corresponding variable from Y.

We will also use the notation

$$[M_1/x_1, \ldots, M_n/x_n]$$

for substitutions of terms M_i for variables x_i in a term. When $\Gamma \cong \Delta$ we use the notation $[\Delta/\Gamma]$ to replace occurrences of variables in a context or term that also occur in Γ with their correspondents (under the chosen bijection) from Δ.

2.1. *Reductions*

The usual rules for $\beta\eta\zeta$-conversions for $\alpha\lambda$ are retained. They are

$$\begin{array}{clcl}
(\beta I) & \text{let } I \text{ be } I \text{ in } N & \to_\beta & N \\
(\eta I) & \text{let } I \text{ be } M \text{ in } I & \to_\eta & M \\
(\beta \wedge 1) & \pi_1\langle M_1, M_2\rangle & \to_\beta & M_1 \\
(\beta \wedge 2) & \pi_2\langle M_1, M_2\rangle & \to_\beta & M_2 \\
(\eta\wedge) & \langle \pi_1 M, \pi_2 M\rangle & \to_\eta & M \\
(\beta *) & \text{let } (x,y) \text{ be } M_1 * M_2 \text{ in } N & \to_\beta & N[M_1/x, M_2/y] \\
(\eta *) & \text{let } (x,y) \text{ be } M \text{ in } x * y & \to_\eta & M \\
(\beta \to) & \text{app}(\lambda x : \sigma.N, M) & \to_\beta & N[M/x] \\
(\eta \to) & \lambda x : \sigma.\text{app}(M, x) & \to_\eta & M \\
(\beta -\!\!*) & \text{app}_*(\lambda_* x : \sigma.N, M) & \to_\beta & N[M/x] \\
(\eta -\!\!*) & \lambda_* x : \sigma.\text{app}_*(M, x) & \to_\eta & M
\end{array}$$

and $x \notin fv(M)$ for both $(\eta \to)$ and $(\eta -\!\!*)$. No rules for $\top$ are required. The ζ-rules (commuting conversions) are required for I, $*$ and $\vee$ only, and are the same for $\alpha\lambda$.

In order to state these we use the notion of *context*. In brief, a context is just a term-with-hole. We may substitute terms for the hole, allowing variable capture. We write $C[-]$ for a such a context (with a hole $[-]$). It should always be clear from the setting in which this terminology is used whether we mean such a context or a bunch of variables. More details on contexts for $\alpha\lambda$ can be found in (Pym 2002).

Then

$$\begin{array}{rcl}
C[\text{ let } I \text{ be } M \text{ in } N] & \to_\zeta & \text{let } I \text{ be } M \text{ in } C[N] \\
C[\text{ let } (x,y) \text{ be } M \text{ in } N] & \to_\zeta & \text{let } (x,y) \text{ be } M \text{ in } C[N] \\
C[\text{case } M \text{ of inl}(x) \Rightarrow N_1 \text{ or inr}(y) \Rightarrow N_2] & \to_\zeta & \text{case } M \text{ of inl}(x) \Rightarrow C[N_1] \\
 & & \text{or inr}(y) \Rightarrow C[N_2]
\end{array}$$

where in the second clause we require that C does not bind x, y.

In addition, we have four $\beta\eta$-conversions for polymorphic quantifiers,

$$\begin{array}{crcl}
(\beta\forall) & \text{App}(\Lambda\alpha.M, \sigma) & \to_\beta & M[\sigma/\alpha] \\
(\beta\forall_*) & \text{App}_*(\Lambda_*\alpha.M, A, \sigma) & \to_\beta & M[\sigma/\alpha] \\
(\eta\forall) & \Lambda\alpha.\text{App}(M, \alpha) : \tau & \to_\eta & M \\
(\eta\forall_*) & \Lambda_*\alpha.\text{App}_*(M, \{\alpha\}, \alpha) : \tau & \to_\eta & M
\end{array}$$

where these terms are all typed over the same hub X and context Γ such that α is not free in Γ. Let $\twoheadrightarrow$ be the reduction relation generated by the single step conversions. These relations give rise to a system of $\beta\eta\zeta$-equalities, namely that generated by taking two terms to be equal when there is a reduction of (either) one to the other.

3. Metatheory

Many of the standard properties (substitution, subject-reduction, normalization) of a lambda calculus hold for $\alpha2\lambda2$. The proofs are refinements of the proofs for $\alpha\lambda$ and $\lambda2$. We begin with a couple of strengthening lemmas.

Lemma 2. If $X \mid \Gamma(\Delta\ ;\ x : \sigma) \vdash M : \tau$ and x is not free in M then $X \mid \Gamma(\Delta) \vdash M : \tau$.

Proof.
The proof is essentially as for $\alpha\lambda$, that is, by induction on derivation of the given judgement. The essential point to observe is that the variable x must have been added spuriously using weakening. The rules for polymorphism add no real complications. As usual, the multinary version is proved in order to deal with contraction (C). □

Now whenever $X \mid \Gamma \vdash M : \tau$ is derivable then $\mathit{ftv}(\tau) \subseteq \mathit{ftv}(M) \subseteq \mathit{ftv}(X)$ and $\mathit{ftv}(\Gamma) \subseteq \mathit{ftv}(X)$ hold. However, it is not always the case that $\mathit{ftv}(\tau) = \mathit{ftv}(M)$ holds.

Suppose that there is a derivation D of $X \mid \Delta \vdash M : \tau$ and that X is a sub-bunch of Y. Then we say that Γ *may be recovered from* Δ *by weakenings over* Y if a derivation of $Y \mid \Gamma \vdash M : \tau$ may be produced by appending some sequence of weakenings — $(W2)$ and (W) — to the root of D. Note that this definition is really independent of D, M and τ, since it is the same as saying that, using ';' only, we can append to Δ all variables of $\mathit{fv}(\Gamma) \setminus \mathit{fv}(\Delta)$ to recover Γ, and have Γ well-formed over Y. We have the following strengthening lemma for type variables.

Lemma 3. If $X(Z) \mid \Gamma \vdash M : \tau$ is derivable, $\alpha \notin \mathit{ftv}(M)$ and either $Z = Y;\alpha$ or $Z = Y, \alpha$, then there is a context Γ' such that $X[Y/Z] \mid \Gamma' \vdash M : \tau$ is derivable, $\alpha \notin \mathit{ftv}(\Gamma')$, and Γ can be recovered from Γ' by weakenings over $X(Z)$. Furthermore, if $\alpha \notin \mathit{ftv}(\Gamma)$ then $\Gamma' = \Gamma$.

Proof. If $\alpha \notin \mathit{ftv}(M)$ then there are no (free or bound) variables in M which have types involving free occurrences of α. Therefore any variables $x : \sigma(\alpha)$ occurring in Γ have been inserted by weakenings (W) in the derivation of $X \mid \Gamma \vdash M : \tau$. Hence, by Lemma 2, we can replace this derivation with a derivation of $X \mid \Gamma' \vdash M : \tau$ in which those extraneous variables are never inserted. Since α does not feature in any of Γ', M or τ, it can only have been introduced into X through some weakening using $(W2)$. We may therefore derive $X[Y/Z] \mid \Gamma' \vdash M : \tau$ by not inserting α at any such weakening. □

The fact that hubs are affine yields an admissible substitution rule for terms.

Proposition 1. (Substitution)
If $X \mid \Gamma(x : \sigma) \vdash N : \tau$ and $X \mid \Delta \vdash M : \sigma$ are derivable and $\mu(N) \cap \mathit{ftv}(M) = \emptyset$ then $X \mid \Gamma[\Delta/x] \vdash N[M/x] : \tau$ is derivable.

Notice that substitution takes place over a fixed hub. To see the necessity of the substitution side-condition (†) observe that $\alpha, \beta \mid x : \alpha \vdash \mathrm{App}_*(\Lambda_*\gamma.x, \{\beta\}, \beta) : \alpha$ and

$\alpha, \beta \mid y : \beta \to \alpha; z : \beta \vdash \mathrm{app}(y, z) : \alpha$ are both derivable but that $\alpha, \beta \mid y : \beta \to \alpha; z : \beta \vdash \mathrm{App}_*(\Lambda_*\gamma.\mathrm{app}(y, z), \{\beta\}, \beta) : \alpha$ is not since the formation of $\mathrm{app}(y, z)$ requires β. The side-condition $(\dagger)$ is used in clause $(\forall_* E)$ of the proof in conjunction with Lemma 3. Notice that weakening around ',' in a hub via $(W2)$ is used in clauses $(\forall_* I)$ and $(\forall_* E)$ of the proof.

Proof. Let $\Gamma[(\Delta_i/\Delta_i')_{1\leq i\leq n}]$ and $N[(M_i/x_i)_{1\leq i\leq n}]$ be the respective notations for n simultaneous substitutions into contexts and terms. The proof of admissibility of substitution is done for the multinary version

$$\frac{X \mid \Gamma(x_1 : \sigma_1 \mid \ldots \mid x_n : \sigma_n) \vdash N : \tau \quad (X \mid \Delta_i \vdash M_i : \sigma_i)_{1\leq i\leq n}}{X \mid \Gamma[(\Delta_i/x_i)_{1\leq i\leq n}] \vdash N[(M_i/x_i)_{1\leq i\leq n}] : \tau} \quad (\dagger_1)\ldots(\dagger_n)$$

for all n, in order to deal with contraction. Each $(\dagger_i)$ says $\mu(N) \cap \mathit{ftv}(M_i) = \emptyset$.

We use induction on derivations, by considering cases of the last rule (r) used in the derivation of the left premise of the substitution. Note that the induction hypothesis is that the rule is admissible for all substitutions into all derivations of lesser height and for all n.

All the cases which introduce or eliminate a connective other than a multiplicative quantifier are straightforward as are the structural rules. We simply push the substitution towards the leaves, splitting it as appropriate at rules with two premises.

$(\forall_* I)$. For simplicity, we present the case with $n = 1$, that is, where we substitute for just one variable. The version for arbitrary finite n is no more difficult. Suppose we have a derivation ending with

$$\frac{X, \alpha \mid \Gamma(x : \sigma) \vdash N : \tau}{X \mid \Gamma(x : \sigma) \vdash \Lambda_*\alpha.N : \forall_*\alpha.\tau}$$

where α is not free in Γ. For any given substitution premise

$$X \mid \Delta \vdash M : \sigma$$

we can weaken via $(W2)$ to form

$$X, \alpha \mid \Delta \vdash M : \sigma$$

then apply the induction hypothesis to get

$$X, \alpha \mid \Gamma[\Delta/x] \vdash N[M/x] : \tau$$

and, finally, since α is not free in Δ, we may use $(\forall_* I)$ to get

$$X \mid \Gamma[\Delta/x] \vdash \Lambda_*\alpha.N[M/x] : \forall_*\alpha.\tau$$

as required.

$(\forall_* E)$. Suppose we have a derivation ending with

$$\frac{Y \mid \Gamma(x : \sigma) \vdash N : \forall_*\alpha.\tau}{Y, Z \mid \Gamma(x : \sigma) \vdash \mathrm{App}_*(N, \mathit{ftv}(Z), \rho) : \tau[\rho/\alpha]} \quad (Z \vdash \rho)$$

for the unary substitution case. The matching substitution premise is of the form

$$Y, Z \mid \Delta \vdash M : \sigma$$

with

$$\mu(\mathrm{App}_*(N, \mathit{ftv}(Z), \rho)) \cap \mathit{ftv}(M) = \emptyset$$

as side-condition. This gives

$$\mathit{ftv}(Z) \cap \mathit{ftv}(M) = \emptyset$$

in particular. Then, by repeated application of Lemma 3, there is some Δ' such that

$$Y \mid \Delta' \vdash M : \sigma$$

is derivable, and Δ may be recovered from Δ' by weakenings over Y, Z.

The condition

$$\mu(N) \cap \mathit{ftv}(M) = \emptyset$$

is satisfied, so the induction hypothesis gives

$$Y \mid \Gamma[\Delta'/x] \vdash N[M/x] : \forall_* \alpha.\tau$$

derivable. From this it follows by $(W2)$ that

$$Y, Z \mid \Gamma[\Delta'/x] \vdash \mathrm{App}_*(N[M/x], \mathit{ftv}(Z), \rho) : \tau[\rho/\alpha]$$

by an application of the $(\forall_* E)$ rule, and

$$Y, Z \mid \Gamma[\Delta'/x] \vdash (\mathrm{App}_*(N, \mathit{ftv}(Z), \rho))[M/x] : \tau[\rho/\alpha]$$

is just the same term. Finally, we can weaken using (W) to give

$$Y, Z \mid \Gamma[\Delta/x] \vdash (\mathrm{App}_*(N, \mathit{ftv}(Z), \rho))[M/x] : \tau[\rho/\alpha]$$

as required. □

There is a substitution property for types.

Proposition 2. If $Y \mid \Gamma \vdash M : \tau$ is derivable and $s : X \longrightarrow Y$ is a substitution of types then $X \mid s^*(\Gamma) \vdash s^*(M) : s^*(\tau)$ is derivable.

Proof. The proof is by induction on the derivation of $Y \mid \Gamma \vdash M : \tau$. Most of the cases are straightforward. The rule $(C2)$ means we really prove the multinary simultaneous substitution to get a strong enough induction hypothesis. The structural rules require a number of simple results indicating the compatibility of substitution with the relations $\cong$ and $\equiv$ on both fibres and contexts. The two most difficult cases are given explicitly below (in unary rather than multinary form).

$(\forall_* I)$. Suppose we have a derivation ending

$$\frac{Y, \alpha \mid \Gamma \vdash M : \tau}{Y \mid \Gamma \vdash \Lambda_* \alpha.M : \forall_* \alpha.\tau} \qquad (\alpha \notin \mathit{ftv}(\Gamma))$$

and a substitution $s = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y$. There is a substitution t of the form $[\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n, \alpha/\alpha] : X, \alpha \longrightarrow Y, \alpha$. The induction hypothesis gives $X, \alpha \mid t^*(\Gamma) \vdash t^*(M) : t^*(\tau)$ and from this we get $X \mid t^*(\Gamma) \vdash \Lambda_* \alpha.t^*(M) : \forall_* \alpha.t^*(\tau)$ from which we may conclude $X \mid s^*(\Gamma) \vdash s^*(\Lambda_* \alpha.M) : s^*(\forall_* \alpha.\tau)$. The $(\forall I)$ case is very similar.

$(\forall_* E)$. Suppose that we have a derivation ending

$$\frac{Y_1 \mid \Gamma \vdash M : \forall_* \alpha.\tau}{Y_1, Y_2 \mid \Gamma \vdash \mathrm{App}_*(M, \mathit{ftv}(Y_2), \phi) : \tau[\phi/\alpha]} \quad (Y_2 \vdash \phi)$$

and that $\mathit{ftv}(Y_1) = \{\beta_1, \ldots, \beta_m\}$ and $\mathit{ftv}(Y_2) = \{\beta_{m+1}, \ldots, \beta_n\}$, where both of these sets may be empty. Further suppose that we have a substitution

$$s : X \longrightarrow Y; [\sigma_1/\beta_1, \ldots, \sigma_m/\beta_m, \sigma_{m+1}/\beta_{m+1}, \ldots, \sigma_n/\beta_n] \ .$$

Let $A_i = \bigcup\{\mathit{ftv}(\sigma_j) \mid \beta_j \in \mathit{ftv}(Y_i)\}$ for $i = 1, 2$. Note that $A_1 \cap A_2 = \emptyset$.

Let $X_i = X\lceil_{A_i}$ for $i = 1, 2$, and consider the sequences $s_1 = [\sigma_1/\beta_1, \ldots, \sigma_m/\beta_m]$ and $s_2 = [\sigma_{m+1}/\beta_{m+1}, \ldots, \sigma_n/\beta_n]$. Now $X_1 \vdash \sigma_j$ for $1 \leq j \leq m$ and $X_2 \vdash \sigma_j$ for $m + 1 \leq j \leq n$.

Fix $i = 1$ or $i = 2$ and suppose $W^1, W^2 \lhd Y_i$. We have $W^1, W^2 \lhd Y_1, Y_2$, so there are V^1 and V^2 with $V^1, V^2 \lhd X$ and $V^k \vdash \sigma_j$ for all $1 \leq j \leq n$ and $k = 1, 2$ such that $\beta_j \in \mathit{ftv}(W^k)$. Define $U^k = V^k\lceil_{X_i}$ for $k = 1, 2$. Then $\beta_j \in W^k$ implies $U^k \vdash \sigma_j$ for $k = 1, 2$ and $1 \leq j \leq n$. Further, $U^1, U^2 \lhd X_i$ since $U^k \lhd X_i$ for $k = 1, 2$ and $U^1, U^2 \lhd X$ and $X_i \lhd X$. Therefore both $s_1 : X_1 \longrightarrow Y_1$ and $s_2 : X_2 \longrightarrow Y_2$ are substitutions.

We then know that $X_2 \vdash s_2^*(\phi)$ and, using the induction hypothesis, we have a derivation of $X_1 \mid s_1^*(\Gamma) \vdash s_1^*(M) : s_1^*(\forall_* \alpha.\tau)$. Since $s_1^*(\forall_* \alpha.\tau) = \forall_* \alpha. s_1^*(\tau)$ we may apply the rule $(\forall_* E)$ to derive

$$X_1, X_2 \mid s_1^*(\Gamma) \vdash \mathrm{App}_*(s_1^*(M), \mathit{ftv}(X_2), s_2^*(\phi)) : (s_1^*(\tau))[s_2^*(\phi)/\alpha] \ .$$

Now this is the same as

$$X_1, X_2 \mid s^*(\Gamma) \vdash s^*(\mathrm{App}_*(M, \mathit{ftv}(Y_2), \phi) : s^*(\tau[\phi/\alpha])$$

and we may apply the rules $(W2)$ and $(E2)$ to give

$$X \mid s^*(\Gamma) \vdash s^*(\mathrm{App}_*(M, \mathit{ftv}(Y_2), \phi) : s^*(\tau[\phi/\alpha])$$

as required.

The additives case $(\forall E)$ follows by a simpler version of this argument. □

There are inversion properties for all four abstractions.

Proposition 3. The four rules below are admissible.

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

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

Proof. The proof for all four rules is similar. We consider only the $\forall_*$ case below: this is the most complicated.

The essential point to observe is that the derivation of $X \mid \Gamma \vdash \Lambda_*\alpha.M : \forall_*\alpha.\tau$ must finish with the rule $(\forall_* I)$ followed a (possibly empty) sequence of uses of rules (W), (C), (E), $(C2)$, $(W2)$, $(E2)$, (TE) and $(\wedge E)$. Let (r) range over these rules. We then use induction on the length, l, of the sequence of rules following the final $(\forall_* I)$. If $l = 0$ then the derivation we want is just the derivation of the premise of $(\forall_* I)$. Now suppose that $l = n + 1$. Suppose that the $n + 1$th rule is an instance of rule (r) with a premise of the form $X' \mid \Gamma' \vdash \Lambda_* \alpha.M' : \forall_* \alpha.\tau'$. A suitable induction hypothesis gives that $X', \alpha \mid \Gamma' \vdash M' : \tau'$. For all cases of (r), rule (r) commutes with $(\forall_* I)$. We may then apply rule (r) to conclude $X, \alpha \mid \Gamma \vdash M : \tau$, as required. □

The propositions above can be used 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.

Proof.
For the proof it is enough to show that each conversion step preserves derivability. This is done in the usual way using the inversion and substitution properties.

The β-reductions for $-\!*$ and $\to$ require only shallow versions of the substitution law, i.e. those in which we substitute for x in a typing context $\Gamma, x : \tau$ or $\Gamma; x : \tau$. However, the β-reduction relation for $*$ requires the deep form of substitution, as given in Proposition 1. The side-condition required for the substitutions is guaranteed in cases $\to$, $-\!*$, $*$ and $\vee$ by the side-conditions (†) in the elimination laws (in fact, that is why those side-conditions are placed on those elimination laws). The cases for the β-reductions for $\forall$ and $\forall_*$ make use of Proposition 2.

The η-reduction cases are proved straightforwardly using inversion and by considering the forms of derivations which are forced by the given redex terms: derivations must (without loss of generality) end with the rules corresponding to the pairing of introduction and elimination in the redex followed by a sequence of structural rules. We can obtain the desired reduced term from a sub-derivation by means of (some of) the same structural rules.

We sketch the proof by giving a number of representative cases.

$(\beta -\!*)$. A derivation of a term $\mathrm{app}_*(\lambda_* x : \sigma.N, M) : \tau$ can be assumed (without loss of generality) to end as follows

$$\dfrac{(-\!*I)\ \dfrac{\dfrac{\vdots}{X \mid \Gamma, x : \sigma \vdash N : \tau}}{X \mid \Gamma \vdash \lambda_* x : \sigma.N : \sigma -\!* \tau} \qquad \dfrac{\vdots}{X \mid \Delta \vdash M : \sigma}}{X \mid \Gamma, \Delta \vdash \mathrm{app}_*(\lambda_* x : \sigma.N, M) : \tau}\ (-\!*E)$$

so we must have $(\mu(\lambda_* x : \sigma.N) \cap \mathit{ftv}(M) = \emptyset)$ in order to apply the final rule. We then have $\mu(N) \cap \mathit{ftv}(M) = \emptyset$ and so we may make a substitution

$$\frac{X \mid \Gamma, x : \sigma \vdash N : \tau \qquad X \mid \Delta \vdash M : \sigma}{X \mid \Gamma, \Delta \vdash N[M/x] : \tau}$$

instead. The argument for case $(\beta \to)$ is almost identical. The arguments for $(\beta *)$ and $(\beta \wedge)$ are similar, except that the full, deep form of substitution is used. In the case of $(\beta \vee)$ we see that both side-conditions on $(\vee E)$ are necessary in order that we may apply the substitution to the appropriate branch.

($\beta\forall_*$). Without loss of generality we have a derivation ending

$$\dfrac{\dfrac{X,\alpha \mid \Gamma \vdash M : \tau}{X \mid \Gamma \vdash \Lambda_*\alpha.M : \forall_*\alpha.\tau}}{X,Y \mid \Gamma \vdash \mathrm{App}_*(\Lambda_*\alpha.M, \mathit{ftv}(Y), \sigma) : \tau[\sigma/\alpha]} \quad (Y \vdash \sigma)$$

where $\alpha \notin \mathit{ftv}(\Gamma)$. By Proposition 2 we have that $X, Y \mid \Gamma \vdash M[\sigma/\alpha] : \tau[\sigma/\alpha]$ is derivable, since $\alpha \notin \mathit{ftv}(\Gamma)$. The case for ($\beta\forall$) is similar.

($\eta \mathrel{-\!\!*}$). Suppose we have a derivation of $X \mid \Gamma \vdash \lambda_* x : \sigma.\mathrm{app}_*(M, x) : \sigma \mathrel{-\!\!*} \tau$. Then, using invertibility, we know that there is a derivation that ends with

$$\dfrac{\dfrac{X \mid \Gamma \vdash M : \sigma \mathrel{-\!\!*} \tau \qquad X \mid x : \sigma \vdash x : \sigma}{X \mid \Gamma, x : \sigma \vdash \mathrm{app}_*(M, x) : \tau}}{X \mid \Gamma \vdash \lambda_* x : \sigma.\mathrm{app}_*(M, x) : \sigma \mathrel{-\!\!*} \tau}$$

without loss of generality. The result is then immediate. The case ($\eta \rightarrow$) is essentially the same.

($\eta\forall_*$). Without loss of generality we have a derivation ending

$$\begin{array}{c}
\vdots \\ \hline
X_1 \mid \Gamma_1 \vdash M_1 : \forall_*\alpha.\tau \\ \hline
X_1, \alpha \mid \Gamma_1 \vdash \mathrm{App}_*(M_1, \{\alpha\}, \alpha) : \tau \\ \hline
X_1 \mid \Gamma_1 \vdash \Lambda_*\alpha.\mathrm{App}_*(M_1, \{\alpha\}, \alpha) : \forall_*\alpha.\tau \\ \hline
\vdots \\ \hline
X \mid \Gamma \vdash \Lambda_*\alpha.\mathrm{App}_*(M, \{\alpha\}, \alpha) : \forall_*\alpha.\tau
\end{array}$$

for some X_1, Γ_1 and M_1. We can then recover $X \mid \Gamma \vdash M : \forall_*\alpha.\tau$ from $X_1 \mid \Gamma_1 \vdash M : \forall_*\alpha.\tau$ by using the rules from the sequence between the abstraction and the root of the original derivation, thus omitting the polymorphic abstraction and application. □

All reductions of the calculus terminate.

Theorem 2. The calculus is strongly normalizing.

Proof. The proof is by the translation method. A similar proof for $\alpha\lambda$ is contained in (Pym 2002). We define recursively a translation of our calculus into the terms of the ordinary polymorphic lambda calculus in the natural way: additive and multiplicative variants of connectives are sent to the same connective in the ordinary polymorphic lambda calculus (for example $*$ is sent to $\wedge$ and Λ_* is sent to Λ). Each one-step reduction of our calculus corresponds to a reduction of the ordinary calculus. Since each reduction sequence of the ordinary calculus is of finite length so must be every reduction sequence of our calculus. □

Confluence (the Church-Rosser property) in $\alpha 2\lambda 2$ has not been studied in detail.

4. Categorical Semantics

The calculus $\alpha 2\lambda 2$ has a categorical semantics that is a hybrid of the indexed category (hyperdoctrine) semantics of $\lambda 2$ with the doubly closed category semantics of $\alpha\lambda$. We omit discussion of disjunction (coproducts) for the remainder of the paper.

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 $\mathbb{B}$: $\otimes$ is the tensor operation, I is its unit, and $r : A \otimes I \longrightarrow A$, $l : I \otimes A \longrightarrow A$, $a_{A,B,C} : (A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C)$, $s_{A,B} : A \otimes B \longrightarrow B \otimes A$ are the components of the right-unit, left-unit, associativity and symmetry natural isomorphisms, respectively.

Let $1_{\mathbb{B}} : \mathbb{B} \longrightarrow \mathbb{B}$ be the identity functor. Let $B : \mathbb{B} \longrightarrow \mathbb{B}$ be the constant functor to any given object $B \in \mathbb{B}$. Recall the definitions of *monoidal functor* and *monoidal natural transformation*. The functors $1_{\mathbb{B}}$, B and $1_{\mathbb{B}} \otimes B$ are all monoidal. The full definitions of symmetric monoidal category, monoidal functor and monoidal natural transformation can all be found in (Mac Lane 1971).

Definition 2. The monoid $\otimes$ is a *pseudoproduct* if there is a monoidal natural transformation $\psi^1_B : 1_{\mathbb{B}} \otimes B \Longrightarrow 1_{\mathbb{B}} : \mathbb{B} \longrightarrow \mathbb{B}$ for every object B in $\mathbb{B}$.

The following lemma provides a useful alternative definition.

Lemma 4. A symmetric monoid $(\otimes, I, a, l, r, s)$ on $\mathbb{B}$ is a pseudoproduct if for every object B in $\mathbb{B}$ there is a (first) *pseudoprojection*, that is, a natural transformation ψ^1_B : $1_{\mathbb{B}} \otimes B \Longrightarrow 1_{\mathbb{B}}$ satisfying the two coherence diagrams given below.

$$\begin{tikzcd}
(A \otimes B) \otimes C \arrow[r, "a"] \arrow[d, "\psi^1"'] & A \otimes (B \otimes C) \arrow[d, "\psi^1"] \arrow[dl, "1_A \otimes \psi^1"'] \\
A \otimes B \arrow[r, "\psi^1"] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \otimes I \arrow[r, "r"] \arrow[d, "1_{A\otimes I}"'] & A \arrow[d, "1_A"] \\
A \otimes I \arrow[r, "\psi^1"] & A
\end{tikzcd}$$

Proof. The proof of this lemma just makes use of the defining diagrams for monoidal functors and natural transformations and the coherence property for symmetric monoidal closed categories. $\square$

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 straightforward 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 $A_\perp$ and functions which preserve the distinguished element $\perp$ has a pseudoproduct given by the coproduct. A pseudoprojection from $A_\perp + B_\perp$ to $A_\perp$ may be taken to be $\perp, b \mapsto \perp,\ a \mapsto a$ for all $a \in A, b \in B$.

Lemma 5. If the unit of a symmetric monoid on $\mathbb{B}$ is the terminal object, that is $I = \top$, then that monoid has precisely one pseudoprojection, and

$$\psi^1_{A,B} = r_A \circ (1_A \otimes !_B)$$

for any objects A and B, where r_A is the component of r at A, and $!_B$ is the unique arrow from B into the terminal object.

Proof. The arrows $\psi^1_{A,B}$ are easily verified to satisfy the conditions of Lemma 4. Furthermore, the left-hand diagram of Lemma 4 together with the coherence conditions for the monoid (in particular, the isomorphism r) force the arrow $\psi^1_{A,B}$ to be given by the above equation. $\square$

A *cartesian doubly closed category* (CDCC) is a category with a pair of symmetric monoidal closed structures, one of which is cartesian. We write the cartesian closed structure as $\times, \rightarrow$ and the other closed structure as $\otimes, \multimap$. 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 functor

$$P : \mathbb{B}^{op} \longrightarrow \mathbb{C}$$

from a base category $\mathbb{B}$ to some category of categories $\mathbb{C}$. For any object B in the base the category $P(B)$ is called the *fibre* over B. The functors $P(s) : P(B') \longrightarrow P(B)$ induced by base arrows $s : B \longrightarrow B'$ are known as *reindexings* or *substitutions*. The notation s^* is often used in place of $P(s)$.

Split indexed categories can be used to model $\lambda 2$ following (Seely 1987). These models are called *hyperdoctrines* and are an extension of the method of modelling quantification of (Lawvere 1969). A detailed account of split indexed categories and hyperdoctrines is given in (Jacobs 1999). Hyperdoctrines for $\lambda 2$ require the fibres to be cartesian closed and their substitutions to preserve this structure appropriately (usually on-the-nose). For quantification over types they also require 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. This corresponds to the substitution of arbitrary types for type variables. In addition, the right-adjoints to certain of the substitutions are required to satisfy the Beck-Chevalley condition.

Definition 3. An $\alpha 2\lambda 2$-*hyperdoctrine* is a split indexed category

$$P : \mathbb{B}^{op} \longrightarrow \mathbf{CDCC}$$

with the following structure and properties:

— a generic object, Ω;
— finite products in $\mathbb{B}$, including terminal object, $\top$;
— a symmetric monoid $(\otimes, I, a, l, r, s)$ in $\mathbb{B}$, with $I = \top$;
— for any projection $\pi : B \times \Omega \longrightarrow B$ in $\mathbb{B}$, the functor $P(\pi)$ has a right-adjoint, Π;
— each $P(\pi)$ satisfies the Beck-Chevalley condition (detailed below);

— for each pseudoprojection $\psi : B \otimes \Omega \longrightarrow B$ induced by the monoid and terminal object, the functor $P(\psi)$ has a right-adjoint, Ψ;
— each $P(\psi)$ satisfies the multiplicative Beck-Chevalley condition (below).

Consider any projection π and pseudoprojection ψ as in Definition 3. Further, suppose that $u : C \longrightarrow B$ is an arrow of $\mathbb{B}$. Let

$$\epsilon : P(\pi) \circ \Pi \Longrightarrow 1_{P(B \times \Omega)} \qquad \text{and} \qquad \zeta : P(\psi) \circ \Psi \Longrightarrow 1_{P(B \otimes \Omega)}$$

be the co-units of the adjunctions $P(\pi) \dashv \Pi$ and $P(\psi) \dashv \Psi$, respectively. For each object τ over $P(B \times \Omega)$, there is a natural transformation

$$m : P(u) \circ \Pi \Longrightarrow \Pi \circ P(u \times 1)$$

called the *(additive) canonical natural transformation.* The transpose of $P(u \times 1)(\epsilon_\tau)$ under the adjunction $P(\pi) \dashv \Pi$,

$$\frac{P(u \times 1)(\epsilon_\tau) : (P(\pi) \circ P(u) \circ \Pi)(\tau) \longrightarrow P(u \times 1)(\tau)}{m_\tau : (P(u) \circ \Pi)(\tau) \longrightarrow (\Pi \circ P(u \times 1))(\tau)} \; ,$$

gives the component, m_τ, at each object τ of $P(B \times \Omega)$. In a similar way, the adjunction $P(\psi) \dashv \Psi$ gives rise to the *(multiplicative) canonical natural transformation*

$$n : P(u) \circ \Psi \Longrightarrow \Psi \circ P(u \otimes 1)$$

using the transpose of $P(u \otimes 1)(\zeta_\tau)$ to give the component n_τ at each τ in $P(B \otimes \Omega)$.

Definition 4. *(The (Additive) Beck-Chevalley Condition).* Let $u : C \longrightarrow B$ be an arrow of $\mathbb{B}$. Then the following square commutes.

$$\begin{tikzcd}
P(B \times \Omega) \arrow[r, "\Pi"] \arrow[d, "P(u \times 1)"'] & P(B) \arrow[d, "P(u)"] \\
P(C \times \Omega) \arrow[r, "\Pi"] & P(C)
\end{tikzcd}$$

and the canonical natural transformation m is the identity.

For multiplicative quantification we have a modified version of the Beck-Chevalley condition. This is the same as the ordinary Beck-Chevalley condition in the case where the monoid is a product.

Definition 5. *(The Multiplicative Beck-Chevalley Condition).* Let $u : C \longrightarrow B$ be an arrow of $\mathbb{B}$. Then the following square commutes.

$$\begin{tikzcd}
P(B \otimes \Omega) \arrow[r, "\Psi"] \arrow[d, "P(u \otimes 1)"'] & P(B) \arrow[d, "P(u)"] \\
P(C \otimes \Omega) \arrow[r, "\Psi"] & P(C)
\end{tikzcd}$$

$$[\![X \vdash T : T]\!] = \top \qquad [\![X \vdash I : I]\!] = I \qquad [\![X(\alpha) \vdash \alpha]\!] = \iota^{-1}_{[\![X]\!]}(\varpi_{X(\alpha)})$$

$$[\![X \vdash \tau \wedge \tau']\!] = [\![X \vdash \tau]\!] \times [\![X \vdash \tau']\!] \qquad [\![X \vdash \tau \to \tau']\!] = [\![X \vdash \tau']\!]^{[\![X \vdash \tau]\!]}$$

$$[\![X \vdash \tau * \tau']\!] = [\![X \vdash \tau]\!] \otimes [\![X \vdash \tau']\!] \qquad [\![X \vdash \tau \mathrel{-\!\!*} \tau']\!] = [\![X \vdash \tau]\!] \multimap [\![X \vdash \tau']\!]$$

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

Fig. 2. Interpretation of Types

and the canonical natural transformation n is the identity.

We turn now to the interpretation of hubs, types, contexts and terms. This is uniquely determined by the structure specified in the $\alpha 2\lambda 2$-hyperdoctrine.

Hubs are interpreted as objects in the base $\mathbb{B}$, with

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

using the objects $\top$, I and operations $\times$ and $\otimes$ in the base.

Types are interpreted as objects $[\![X \vdash \tau]\!]$ of the fibre $P([\![X]\!])$. The interpretation is given in Figure 2. In that figure $\top$, $\times$, $\to$, I, $\otimes$ and $\multimap$ come from the doubly closed structure of the fibre $P([\![X]\!])$, Π is the right-adjoint to $P(\pi)$, induced by $\pi : [\![X]\!] \times \Omega \longrightarrow [\![X]\!]$, and Ψ is the right-adjoint to $P(\psi)$, which comes from $\psi : [\![X]\!] \otimes \Omega \longrightarrow [\![X]\!]$. Finally, $\varpi_{X(\alpha)} : [\![X]\!] \longrightarrow \Omega$ is the arrow in $\mathbb{B}$ that tracks the weakening of α to $X(\alpha)$. That is,

$$\begin{aligned} \varpi_\alpha &= 1_\Omega \\ \varpi_{X(\alpha);X'} &= \varpi_{X(\alpha)} \circ \pi^1 \qquad & \varpi_{X';X(\alpha)} &= \varpi_{X(\alpha)} \circ \pi^2 \\ \varpi_{X(\alpha),X'} &= \varpi_{X(\alpha)} \circ \psi^1 \qquad & \varpi_{X',X(\alpha)} &= \varpi_{X(\alpha)} \circ \psi^2 \end{aligned}$$

using projections $\pi^1 : [\![X(\alpha)]\!] \times [\![X']\!] \longrightarrow [\![X(\alpha)]\!]$ and $\pi^2 : [\![X']\!] \times [\![X(\alpha)]\!] \longrightarrow [\![X(\alpha)]\!]$ and pseudoprojections $\psi^1 : [\![X(\alpha)]\!] \otimes [\![X']\!] \longrightarrow [\![X(\alpha)]\!]$ and $\psi^2 : [\![X']\!] \otimes [\![X(\alpha)]\!] \longrightarrow [\![X(\alpha)]\!]$.

Contexts are interpreted as objects of $P([\![X]\!])$ by extension of the interpretation of types:

$$[\![X \vdash x : \tau]\!] = [\![X \vdash \tau]\!]$$

$$[\![X \vdash \Gamma;\Delta]\!] = [\![X \vdash \Gamma]\!] \times [\![X \vdash \Delta]\!] \qquad [\![X \vdash \Gamma,\Delta]\!] = [\![X \vdash \Gamma]\!] \otimes [\![X \vdash \Delta]\!]$$

where this uses the product and monoidal structure of fibres.

Before we give the semantics of terms we introduce some auxiliary concepts and notations. *Contexts-with-holes* are of the form $X \vdash \Gamma(-)$ and defined recursively by

— $X \vdash (-)$ is a context-with-hole
— if we have the context $X \vdash \Delta$ and the context-with-hole $X \vdash \Gamma(-)$ then

$$X \vdash \Gamma(-);\Delta \qquad X \vdash \Delta;\Gamma(-) \qquad X \vdash \Gamma(-),\Delta \qquad X \vdash \Delta,\Gamma(-)$$

are contexts-with-holes.

Let

$$C_A : P(\llbracket X \rrbracket) \longrightarrow P(\llbracket X \rrbracket)$$

be the constant functor to the object A of the fibre over $\llbracket X \rrbracket$. For any context-with-hole $X \vdash \Gamma(-)$ there is a functor

$$F_{X \vdash \Gamma(-)} : P(\llbracket X \rrbracket) \longrightarrow P(\llbracket X \rrbracket)$$

as given by the recursive definition

— $F_{X \vdash (-)} = id_{P(\llbracket X \rrbracket)}$, the identity functor on $P(\llbracket X \rrbracket)$.
— $F_{X \vdash \Gamma(-);\Delta} = F_{X \vdash \Gamma(-)} \times C_{\llbracket X \vdash \Delta \rrbracket}$
— $F_{X \vdash \Delta;\Gamma(-)} = C_{\llbracket X \vdash \Delta \rrbracket} \times F_{X \vdash \Gamma(-)}$
— $F_{X \vdash \Gamma(-),\Delta} = F_{X \vdash \Gamma(-)} \otimes C_{\llbracket X \vdash \Delta \rrbracket}$
— $F_{X \vdash \Delta,\Gamma(-)} = C_{\llbracket X \vdash \Delta \rrbracket} \otimes F_{X \vdash \Gamma(-)}$

where $\times$ and $\otimes$ are taken pointwise.

In a similar way the *hubs-with-hole* are defined as follows:

— $(-)$ is a hub-with-hole;
— if $X(-)$ is a hub-with-hole then so are

$$X(-);Y \qquad Y;X(-) \qquad X(-),Y \qquad Y,X(-)\ .$$

Given any arrow $m : \llbracket Y \rrbracket \to \llbracket Z \rrbracket$ and any hub-with-hole $X(-)$ there is an arrow

$$m_{X(-)} : \llbracket X(Y) \rrbracket \longrightarrow \llbracket X(Z) \rrbracket$$

defined by:

1 if $X(-) = (-)$ then $m_{X(-)} = m$
2 if $X(-) = X''; (X'(-))$ then $m_{X(-)} = id_{\llbracket X'' \rrbracket} \times m_{X'(-)}$
3 if $X(-) = (X''(-)); X'$ then $m_{X(-)} = m_{X''(-)} \times id_{\llbracket X' \rrbracket}$
4 if $X(-) = X'', (X'(-))$ then $m_{X(-)} = id_{\llbracket X'' \rrbracket} \otimes m_{X'(-)}$
5 if $X(-) = (X''(-)), X'$ then $m_{X(-)} = m_{X''(-)} \otimes id_{\llbracket X' \rrbracket}$.

Terms $X \mid \Gamma \vdash M : \tau$ are interpreted as morphisms

$$\llbracket X \mid \Gamma \vdash M : \tau \rrbracket : \llbracket X \vdash \Gamma \rrbracket \longrightarrow \llbracket X \vdash \tau \rrbracket$$

in $P(\llbracket X \rrbracket)$. The interpretation is given in Figure 3 and Figure 4.

In Figure 3 the arrow π^1 is the first projection, the arrow $i : \llbracket X \vdash \Gamma \rrbracket \longrightarrow \llbracket X \vdash \Delta \rrbracket$ is the isomorphism from the CDCC structure and $\delta : \llbracket X \vdash \Delta \rrbracket \longrightarrow \llbracket X \vdash \Delta \rrbracket \times \llbracket X \vdash \Delta' \rrbracket$ is the diagonal. The operators $(-)^{\wedge}$ and $(-)^{\vee}$ give the transposes of the additive exponential, $(-)^{\frown}$ and $(-)^{\smile}$ give the transposes of the linear exponential. The functors $F_{X \vdash \Gamma(-)}$ are used to lift arrows $m : \llbracket X \vdash \Delta \rrbracket \longrightarrow \llbracket X \vdash \Delta' \rrbracket$ into arrows $F_{X \vdash \Gamma(-)}(m) : \llbracket X \vdash \Gamma(\Delta) \rrbracket \longrightarrow \llbracket X \vdash \Gamma(\Delta') \rrbracket$.

In Figure 4, the operations $(-)^{\vartriangle}$ and $(-)^{\triangledown}$ give the transposes of $P(\pi) \dashv \Pi$, $(-)^{\triangleleft}$ and $(-)^{\triangleright}$ give the transposes of $P(\psi) \dashv \Psi$. The morphism $\iota(B) : \llbracket Y \rrbracket \longrightarrow \Omega$ in the base arises because Ω is generic. The arrow $w : \llbracket Y(X) \rrbracket \longrightarrow \llbracket X \rrbracket$ is weakening (arising from projections and pseudoprojections). The arrow $i : \llbracket Y \rrbracket \longrightarrow \llbracket X \rrbracket$ is the isomorphism induced by $X \equiv Y$. The arrow $\delta : \llbracket X \rrbracket \longrightarrow \llbracket X; X' \rrbracket$ is the diagonal, where $X \cong X'$.

$$\llbracket X \mid x:\tau \vdash x:\tau \rrbracket = 1_{\llbracket X\vdash\tau\rrbracket} \qquad \llbracket X \mid \emptyset \vdash \top:\top \rrbracket = 1_{\top} \qquad \llbracket X \mid \emptyset_* \vdash I:I \rrbracket = 1_{I}$$

$$\frac{\llbracket X \vdash \Gamma(\emptyset) \vdash N:\tau \rrbracket = n \qquad \llbracket X \mid \Delta \vdash M:\top \rrbracket = m}{\llbracket X \mid \Gamma(\Delta) \vdash N[M/\top]:\tau \rrbracket = n \circ F_{X\vdash\Gamma(-)}(m)}$$

$$\frac{\llbracket X \mid \Gamma(\emptyset_*) \vdash N:\tau \rrbracket = n \qquad \llbracket X \mid \Delta \vdash M:I \rrbracket = m}{\llbracket X \mid \Gamma(\Delta) \vdash \text{ let } I \text{ be } M \text{ in } N:\tau \rrbracket = n \circ F_{X\vdash\Gamma(-)}(m)}$$

$$\frac{\llbracket X \mid \Gamma \vdash M:\tau \rrbracket = m \qquad \llbracket X \mid \Delta \vdash N:\tau' \rrbracket = n}{\llbracket X \mid \Gamma;\Delta \vdash \langle M,N\rangle : \tau \wedge \tau' \rrbracket = m \times n}$$

$$\frac{\llbracket X \mid \Gamma \vdash M:\tau \rrbracket = m \qquad \llbracket X \mid \Delta \vdash N:\tau' \rrbracket = n}{\llbracket X \mid \Gamma,\Delta \vdash M * N : \tau * \tau' \rrbracket = m \otimes n}$$

$$\frac{\llbracket X \mid \Gamma \vdash M:\phi\wedge\psi \rrbracket = m \qquad \llbracket X \mid \Gamma;(x:\phi;y:\psi) \vdash N:\tau \rrbracket = n}{\llbracket X \mid \Gamma \vdash N[\pi^1 M/x, \pi^2 M/y]:\tau \rrbracket = n \circ \langle 1_{\llbracket X\vdash\Gamma\rrbracket}, m\rangle}$$

$$\frac{\llbracket X \mid \Gamma(x:\phi,y:\psi) \vdash N:\tau \rrbracket = n \qquad \llbracket X \mid \Delta \vdash M:\phi*\psi \rrbracket = m}{\llbracket X \mid \Gamma(\Delta) \vdash \text{ let } (x,y) \text{ be } M \text{ in } N:\tau \rrbracket = n \circ F_{X\vdash\Gamma(-)}(m)}$$

$$\frac{\llbracket X \mid \Gamma;x:\phi \vdash M:\psi \rrbracket = f}{\llbracket X \mid \Gamma \vdash \lambda x:\phi.M : \phi \to \psi \rrbracket = f^{\wedge}}$$

$$\frac{\llbracket X \mid \Gamma,x:\phi \vdash M:\psi \rrbracket = f}{\llbracket X \mid \Gamma \vdash \lambda_* x:\phi.M : \phi \mathrel{-\!\!*} \psi \rrbracket = f^{\frown}}$$

$$\frac{\llbracket X \mid \Gamma \vdash M:\phi\to\psi \rrbracket = m \qquad \llbracket X \mid \Delta \vdash N:\phi \rrbracket = n}{\llbracket X \mid \Gamma;\Delta \vdash \mathrm{app}(M,N):\psi \rrbracket = m^{\vee} \circ (1_{\llbracket X\vdash\Gamma\rrbracket} \times n)}$$

$$\frac{\llbracket X \mid \Gamma \vdash M:\phi \mathrel{-\!\!*} \psi \rrbracket = m \qquad \llbracket X \mid \Delta \vdash N:\phi \rrbracket = n}{\llbracket X \mid \Gamma,\Delta \vdash \mathrm{app}_*(M,N):\psi \rrbracket = m^{\smile} \circ (1_{\llbracket X\vdash\Gamma\rrbracket} \otimes n)}$$

$$\frac{\llbracket X \mid \Gamma(\Delta) \vdash M:\tau \rrbracket = m}{\llbracket X \mid \Gamma(\Delta;\Delta') \vdash M:\tau \rrbracket = m \circ F_{X\vdash\Gamma(-)}(\pi^1)} \qquad \frac{\llbracket X \mid \Gamma \vdash M:\tau \rrbracket = m}{\llbracket X \mid \Delta \vdash M:\tau \rrbracket = m \circ i}\ (\Gamma \equiv \Delta)$$

$$\frac{\llbracket X \mid \Gamma(\Delta;\Delta') \vdash M:\tau \rrbracket = m}{\llbracket X \mid \Gamma(\Delta) \vdash (M:\tau)[\Delta/\Delta'] \rrbracket = m \circ F_{X\vdash\Gamma(-)}(\delta)}\ (\Delta \cong \Delta')$$

Fig. 3. Interpretation of Terms I

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

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

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

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

$$\frac{[\![X \mid \Gamma \vdash M : \tau]\!] = m}{[\![Y(X) \mid \Gamma \vdash M : \tau]\!] = P(w)(m)} \qquad \frac{[\![X \mid \Gamma \vdash M : \tau]\!] = m}{[\![Y \mid \Gamma \vdash M : \tau]\!] = P(i)(m)} \ (X \equiv Y)$$

$$\frac{[\![Y(X;X') \mid \Gamma \vdash M : \tau]\!] = m}{[\![Y(X) \mid \Gamma[X/X'] \vdash (M:\tau)[X/X']]\!] = P(\delta_{Y(-)})(m)} \ (X \cong X')$$

Fig. 4. Interpretation of Terms II

4.1. *Soundness*

We have shown above that every well-formed term $X \mid \Gamma \vdash M : \tau$ can be interpreted as an arrow $[\![X \vdash \Gamma]\!] \longrightarrow [\![X \vdash \tau]\!]$ in $P([\![X]\!])$.

The substitution operations for terms and types may be interpreted. We indicate this for (unary) substitution of a type in another type and for a term in another term. As usual, term substitution corresponds (essentially) to composition whilst type substitution makes use of the generic object. The substitution of a term for a variable takes place in a fixed hub, and its interpretation is modelled in the corresponding CDCC as in (Pym 2002).

Proposition 4. (Substitution)
The operations of type and term substitution have interpretations in P. Furthermore, these agree with the interpretations of substitutions in the syntax.

1 A substitution $s = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y$ gives a functor $[\![s]\!] : P([\![Y]\!]) \longrightarrow P([\![X]\!])$. In the particular case $s = [\sigma/\beta]$ the generic object gives $[\![s]\!] = P(\iota(A)) : P(\Omega) \longrightarrow P([\![X]\!])$, where $[\![X \vdash \sigma]\!] = A \in P([\![X]\!])$. If $[\![Y(\beta) \vdash \tau]\!] = B \in P([\![Y]\!])$ and $[\![X \vdash \sigma]\!] = A \in P([\![X]\!])$ then

$$[\![Y(X) \vdash \tau[\sigma/\beta]]\!] = P((\iota(A))_{Y(-)})(B)\ .$$

2 If $[\![X \mid \Gamma(x:\tau) \vdash N : \sigma]\!] = n : [\![X \vdash \Gamma(x:\tau)]\!] \longrightarrow [\![X \vdash \sigma]\!]$ and $[\![X \mid \Delta \vdash M : \tau]\!] = m : [\![X \vdash \Delta]\!] \longrightarrow [\![X \vdash \tau]\!]$ and $\mu(N) \cap \mathit{ftv}(M) = \emptyset$, then $[\![X \mid \Gamma(\Delta) \vdash N[M/x] : \sigma]\!]$ is given by the arrow

$$[\![X \vdash \Gamma(\Delta)]\!] \xrightarrow{F_{X\vdash\Gamma(-)}(m)} [\![X \vdash \Gamma(x:\tau)]\!] \xrightarrow{n} [\![X \vdash \sigma]\!]$$

over $[\![X]\!]$.

Proof. The proof of the first is an induction on the structure of types. The proof of the second of these is a lengthy induction, that consists of pushing substitutions to the leaves of derivations. $\square$

The equalities that arise from reductions are sound in the model.

Proposition 5. (Equational Soundness) If an equality $X \mid \Gamma \vdash M = M' : \tau$ is derivable then $[\![X \mid \Gamma \vdash M : \tau]\!] = [\![X \mid \Gamma \vdash M' : \tau]\!]$ holds in $P([\![X]\!])$.

Proof. The syntactic equalities are generated by the $\beta\eta\zeta$-conversions and all of these take place over a fixed hub, except for the reductions for the quantifiers. We know that the equalities over any hub (which are just the equalities for $\alpha\lambda$) are all validated in the corresponding CDCC.

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 equations hold because the indexed category is split. The equalities for additive quantification follow by the obvious modifications. $\square$

We have not explicitly given a coherence theorem for interpretations. Strictly speaking, the interpretations above are defined for derivations of sequents; we have not shown that all interpretations of the same sequent are equal, and we shall not do so here. We are not aware of a detailed coherence proof for the interpretation of $\alpha\lambda$ in CDCCs either, although it is suggested by both (Pym 2002) and (O'Hearn 2003) that this is unproblematic. Supposing this to be the case, we also believe that the coherence property holds for our hyperdoctrine interpretation of bunched polymorphism. This is in view of the Beck-Chevalley conditions, the soundness of interpretation of substitution and the soundness of syntactic equality. We can see no reason why the standard categorical, cut-elimination-like, method cannot be used to complete the proof.

4.2. *Completeness*

We now turn our attention to the issue of completeness of the hyperdoctrine interpretation. We prove this by the usual method, following the construction for $\lambda 2$-hyperdoctrines. That is, we build a generic (term) model from the syntax such that if an equation holds between interpreted terms then it must also hold in the theory.

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

The congruence $\cong$ on hubs extends to well-formed types, type substitutions and terms:

$$\begin{array}{rcl}
(X \vdash \sigma) \cong (Y \vdash \tau) & \Longleftrightarrow & X \cong Y \ \&\ \tau = \sigma[Y/X] \\[2ex]
(s = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y) & \Longleftrightarrow & X \cong X' \ \&\ Y \cong Y' \ \& \\
\cong \ (t = [\tau_1/\beta'_1, \ldots, \tau_n/\beta'_n] : X' \longrightarrow Y') & & (X \vdash \sigma_i) \cong (X' \vdash \tau_i) \\[2ex]
(X \mid \Delta \vdash M : \sigma) \cong (Y \mid \Gamma \vdash N : \tau) & \Longleftrightarrow & X \cong Y \ \&\ \Gamma = \Delta[Y/X] \ \& \\
& & N = M[Y/X] \ \&\ \tau = \sigma[Y/X]
\end{array}$$

for all hubs X and Y. Again, we will tend to use representatives for equivalence classes in what follows and will often just write τ, for example, rather than $X \vdash \tau$. Taking quotients as above gives a version of Lemma 1.

Definition 6. The category $\mathbb{B}$ has objects consisting of equivalence classes of hubs and arrows consisting of equivalence classes of type substitutions.

It is a straightforward matter to verify that the category is well-defined, in particular that $\cong$ is a congruence for composition. It is also straightforward to verify that the category has the structure required for the base of an $\alpha 2\lambda 2$-hyperdoctrine. The category has a product: the product of X and Y is given by $X; Y$ (renaming as appropriate so that X and Y are disjoint), the first projection is given by the substitution $\pi : [\alpha_1/\alpha_1, \ldots, \alpha_m/\alpha_m] : X; Y \longrightarrow X$, where $\mathit{ftv}(X) = \{\alpha_1, \ldots, \alpha_m\}$ and the second projection is given in the evident way. There is also a diagonal map $\delta_X : X \longrightarrow X; X'$ whenever $X \cong X'$, given by $[\alpha_1/\alpha_1, \ldots, \alpha_m/\alpha_m, \alpha_1/\alpha'_1, \ldots, \alpha_m/\alpha'_m]$. In a similar way, there is also a symmetric monoid: the monoidal product of X and Y is X, Y and the unit and symmetry isomorphisms are the evident ones. The pseudoprojection is given by $\psi : [\alpha_1/\alpha_1, \ldots, \alpha_m/\alpha_m] : X, Y \longrightarrow X$, where $\mathit{ftv}(X) = \{\alpha_1, \ldots, \alpha_m\}$. However, there is no analogue of the diagonal map above because of the disjointness condition on type substitutions.

Lemma 6. $\mathbb{B}$ has finite products and a symmetric monoid which is a pseudoproduct.

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 (Pym 2002). *Objects* are equivalence classes of types τ that are well-formed over X under the equivalence relation $\cong$. A *morphism* from σ to τ is an equivalence class of term formations $X \mid x : \sigma \vdash M : \tau$, and we write such an arrow as a pair (x, M). The equivalence is generated by α-equality for variables, the $\beta\eta\zeta$-rules of $\alpha 2\lambda 2$ and the congruence $\cong$ on hubs extended to terms. Taking equivalence classes with respect to $\cong$ ensures that the objects and morphisms of fibres are well-defined. Composition of morphisms is achieved through substitution: α-conversion is used to rename variables and type variables of representatives as necessary, thus avoiding unwanted variable capture and ensuring that the side-condition (on the free type variables, see Proposition 1) for substitutions is achieved.

Lemma 7. Each fibre $P(X)$ is a CDCC.

Every arrow $u : X \longrightarrow Y$ of $\mathbb{B}$ yields a substitution functor $P(u) : P(Y) \longrightarrow P(X)$

between fibres. The functor is just substitution of types (up to equivalence classes under $\cong$). It is defined by

$$\begin{aligned} P(u)(\tau) &= u^*(\tau) \\ P(u)((x,M)) &= (y, u^*(M[y/x])) : u^*(\sigma) \longrightarrow u^*(\tau) \end{aligned}$$

for any type τ and arrow $(x, M) : \sigma \longrightarrow \tau$ in $P(Y)$, and any $y : \forall_* \alpha.\sigma$ that is not bound by substitution for x in M. Both the object and arrow assignments can be verified to be well-defined and calculations can be performed to show that $P(u)$ is indeed functorial. Further calculations (which rest upon simple properties of type substitutions) show that the functors $P(u)$ preserve the CDCC structure on-the-nose.

Lemma 8. Each arrow $u : X \longrightarrow Y$ of $\mathbb{B}$ yields a substitution functor $P(u) : P(Y) \longrightarrow P(X)$ that preserves the chosen CDCC structure on-the-nose.

The identity arrow in the base category corresponds to the identity substitution. Considering the identity $1_Z : Z \longrightarrow Z$ or the composite of two arrows

$$X \xrightarrow{u} Y \xrightarrow{v} Z$$

in $\mathbb{B}$ we have that

$$P(1_Z)(\tau) = (1_Z)^*(\tau) = \tau$$

$$P(u)(P(v)(\tau)) = u^*(v^*(\tau)) = (v \circ u)^*(\tau) = P(v \circ u)(\tau)$$

for any type τ in $P(Z)$. A similar calculation for arrows shows that P defines a (contravariant) functor out of $\mathbb{B}$.

Lemma 9. The object and arrow assignments above constitute a functor $P : \mathbb{B}^{op} \longrightarrow$ **CDCC**.

The projection and pseudoprojection

$$\pi : X \times \Omega \longrightarrow X \qquad \text{and} \qquad \psi : X \otimes \Omega \longrightarrow X$$

give rise to *weakenings*. These are functors

$$P(\pi) : P(X) \longrightarrow P(X \times \Omega) \qquad \text{and} \qquad P(\psi) : P(X) \longrightarrow P(X \otimes \Omega)$$

respectively, that are inclusions on both objects and arrows.

Lemma 10. The functors $P(\pi)$ and $P(\psi)$ have right-adjoints

$$\Pi : P(X \times \Omega) \longrightarrow P(X) \qquad \text{and} \qquad \Psi : P(X \otimes \Omega) \longrightarrow P(X)$$

respectively.

Proof. We only give the proof for pseudoprojection. The proof for projection is a trivial modification.

The right-adjoint

$$\Psi : P(X \otimes \Omega) \longrightarrow P(X)$$

is defined so that

$$\begin{array}{rcl}\Psi(\tau) & = & \forall_* \alpha.\tau \\ \Psi((x, M)) & = & (y, \Lambda_* \alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x])\end{array}$$

for all objects τ and arrows $(x, M) : \sigma \longrightarrow \tau$ over X, α. The choice of variable y is irrelevant, so long as it has type $\forall_* \alpha.\sigma$.

Consider the identity morphism $(x, x) : \tau \longrightarrow \tau$ over X. Now

$$\begin{array}{rcl}\Psi((x, x)) & = & (y, \Lambda_* \alpha.(x[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x]) \\ & = & (y, \Lambda_* \alpha.\mathrm{App}_*(y, \{\alpha\}, \alpha)) \\ & = & (y, y)\end{array}$$

using the η-rules. So Ψ maps identity arrows to identity arrows.

Consider any two arrows $(x, M) : \sigma \longrightarrow \tau$ and $(z, N) : \tau \longrightarrow \upsilon$ over X. These have composite $(x, N[M/z]) : \sigma \longrightarrow \upsilon$. Now

$$\begin{array}{rcl}\Psi((z, N) \circ (x, M))) & = & \Psi(x, N[M/z]) \\ & = & (y, \Lambda_* \alpha.(N[M/z])[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x]) \\ & = & (y, \Lambda_* \alpha.N[M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x]/z])\end{array}$$

holds, using an η-rule. On the other hand,

$$\begin{aligned}\Psi(z, N) \circ \Psi(x, M) &= (w, \Lambda_* \alpha.N[\mathrm{App}_*(w, \{\alpha\}, \alpha)/z]) \circ (y, \Lambda_* \alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x]) \\ &= (y, (\Lambda_* \alpha.N[\mathrm{App}_*(w, \{\alpha\}, \alpha)/z])[(\Lambda_* \alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x])/w] \\ &= (y, \Lambda_* \alpha.N[\mathrm{App}_*(\Lambda_* \alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x], \{\alpha\}, \alpha)/z]) \\ &= (y, \Lambda_* \alpha.N[M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x]/z])\end{aligned}$$

holds, using a β-rule. Therefore Ψ is functorial.

There is a bijection

$$\frac{\sigma = P(\psi)(\sigma) \longrightarrow \tau}{\sigma \longrightarrow \Psi(\tau) = \forall_* \alpha.\tau}$$

between arrows, and this is natural in σ over Ω and τ over $X \otimes \Omega$.

Define, for any σ and τ, the function

$$f_{\sigma,\tau} : P(X \otimes \Omega)(P(\psi)(\sigma), \tau) \longrightarrow P(X)(\sigma, \Psi(\tau))$$

by

$$f_{\sigma,\tau}(x, M) = (x, \Lambda_* \alpha.M)$$

for all arrows (x, M) of $P(X \otimes \Omega)$. Define the inverse

$$f^{-1}_{\sigma,\tau} : P(X)(\sigma, \Psi(\tau)) \longrightarrow P(X \otimes \Omega)(P(\psi)(\sigma), \tau)$$

so that

$$f^{-1}_{\sigma,\tau}(x, N) = (x, \mathrm{App}_*(N, \{\alpha\}, \alpha))$$

for every arrow (x, N) of $P(X)$. The $\beta\eta$-rules for Λ_* show $f_{\sigma,\tau}$ to be a bijection.

Naturality of the bijection is established using a β-rule as follows. Let $(y, N) : \sigma' \longrightarrow \sigma$

be an arrow over X and $(z,Q) : \tau \longrightarrow \tau'$ be an arrow over $X \otimes \Omega$. Let

$$\beta = P(X)((y,N), \Psi(z,Q)) : P(X)(\sigma, \Psi(\tau)) \longrightarrow P(X)(\sigma', \Psi(\tau'))$$

$$\alpha = P(X \otimes \Omega)(P(\psi)(y,N), (z,Q)) : P(X \otimes \Omega)(P(\psi)(\sigma), \tau) \longrightarrow P(X \otimes \Omega)(P(\psi)(\sigma'), \tau')$$

be the evident functions induced using the hom-functors. The commutativity of the square

$$\begin{tikzcd}
P(X \otimes \Omega)(P(\psi)(\sigma), \tau) \arrow[r, "f_{\sigma,\tau}"] \arrow[d, "\alpha"'] & P(X)(\sigma, \Psi(\tau)) \arrow[d, "\beta"] \\
P(X \otimes \Omega)(P(\psi)(\sigma'), \tau') \arrow[r, "f_{\sigma',\tau'}"'] & P(X)(\sigma', \Psi(\tau'))
\end{tikzcd}$$

for $f_{\sigma,\tau}$ is established by the calculations below.

Let $(x, M) : P(\psi)(\sigma) \longrightarrow \tau$ be an arrow in $P(X \otimes \Omega)$. We have

$$\begin{array}{rcl}
(\beta \circ f_{\sigma,\tau})(x,M) & = & \beta(x, \Lambda_*\alpha.M) \\
& = & (w, \Lambda_*\alpha.Q[\mathrm{App}_*(w, \{\alpha\}, \alpha)/z]) \circ (x, \Lambda_*\alpha.M) \circ (y,N) \\
& = & (w, \Lambda_*\alpha.Q[\mathrm{App}_*(w, \{\alpha\}, \alpha)/z]) \circ (y, \Lambda_*\alpha.M[N/x]) \\
& = & (y, \Lambda_*\alpha.Q[(\mathrm{App}_*(\Lambda_*\alpha.M[N/x], \{\alpha\}, \alpha))/z]) \\
& = & (y, \Lambda_*\alpha.Q[(M[N/x])/z])
\end{array}$$

where $w : \forall_*\alpha.\tau$ and the final step is by a β-rule. On the other hand,

$$\begin{array}{rcl}
(f_{\sigma',\tau'} \circ \alpha)(x,M) & = & f_{\sigma',\tau'}((z,Q) \circ (x,M) \circ (y,N)) \\
& = & f_{\sigma',\tau'}((y, Q[(M[N/x])/z])) \\
& = & (y, \Lambda_*\alpha.Q[(M[N/x])/z])
\end{array}$$

and this completes the proof.

□

Lemma 11. The Beck-Chevalley conditions hold for Π and Ψ.

Proof. We give the proof for Ψ only. The proof for Π is almost identical. Given a substitution $u = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y$ we perform a couple of straightforward calculations. Note that there is a substitution $u \otimes 1 = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n, \alpha/\alpha] : X, \alpha \longrightarrow Y, \alpha$, which we usually write as $[\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n]$. We have

$$(P(u) \circ \Psi)(\tau) = u^*(\forall_*\alpha.\tau) = \forall_*\alpha.((u \otimes 1)^*\tau) = (\Psi \circ P(u \otimes 1))(\tau)$$

for any object τ of $P(Y \otimes \Omega)$. Also,

$$\begin{array}{rcl}
(P(u) \circ \Psi)((x, M)) & = & u^*(\Psi((x, M))) \\
& = & u^*((y, \Lambda_*\alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x])) \\
& = & (z, (\Lambda_*\alpha.M[\mathrm{App}_*(y, \{\alpha\}, \alpha)/x])[z/y][\sigma_1/\beta_1, \ldots \sigma_n/\beta_n]) \\
& = & (z, (\Lambda_*\alpha.M[\mathrm{App}_*(z, \{\alpha\}, \alpha)/x])[\sigma_1/\beta_1, \ldots \sigma_n/\beta_n]) \\
& = & (z, \Lambda_*\alpha.((M[w/x][\sigma_1/\beta_1, \ldots \sigma_n/\beta_n])[\mathrm{App}_*(z, \{\alpha\}, \alpha)/w])) \\
& = & \Psi((w, M[w/x][\sigma_1/\beta_1, \ldots \sigma_n/\beta_n])) \\
& = & \Psi((u \otimes 1)^*((x, M))) \\
& = & (\Psi \circ P(u \otimes 1))((x, M))
\end{array}$$

for any arrow $(x, M) : \sigma \longrightarrow \tau$ in $P(Y \otimes \Omega)$ and any variables $y : \forall_*\alpha.\sigma$ in $P(Y)$, $z : \forall_*\alpha.\sigma[\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n]$ in $P(X)$ and $w : \sigma[\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n]$ in $P(X \otimes \Omega)$ that are not bound by the above substitutions.

The evident calculations show that the additive and multiplicative canonical natural transformations are both identities. $\square$

An arrow from X to Ω in $\mathbb{B}$ simply consists of a type τ such that $X \vdash \tau$. Hence, the identity gives a natural bijection between the hom-sets $\mathbb{B}(X, \Omega)$ and fibres $P(X)$.

Lemma 12. The indexed category P has a generic object Ω.

Putting all of the above information together we have established that P has the required structure for a model.

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

The completeness theorem is an immediate corollary, since P is constructed from the syntax and each term is interpreted by its own $\alpha\beta\eta\zeta$-equivalence class.

Theorem 4. (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. Hyperdoctrine Structure on Partial Equivalence Relations

Partial equivalence relations on the natural numbers give rise to one of the simplest and most elegant models of the polymorphic lambda calculus. This was shown early-on by (Girard 1972) but has been much studied since. Building upon this we show how the category of partial equivalence relations supports a model for additive polymorphism and first-order bunching. As usual this is structured as a suitable hyperdoctrine. We exhibit a system of pseudoprojections and adjoints between the fibres that is distinct from those used to model additive polymorphism. The idea is to take multiplicatively combined type variables to refer to disjoint PERs, thus providing a disjointness reading for multiplicative polymorphism. However, the system of adjoints (for the pseudoprojections) fails to satisfy the required Beck-Chevalley condition, for which we give an example.

5.1. *The Category of PERs*

A *partial equivalence relation*, PER for short, consists of a symmetric, transitive, binary relation $R \subseteq \mathbb{N} \times \mathbb{N}$ on the natural numbers. Define the *domain* of R to be the set

$$dom(R) = \{n \in \mathbb{N} \mid nRn\}$$

and write

$$dom(R)/R = \{[n] \mid n \in dom(R)\}$$

for the set of (non-empty) equivalence classes of R.

Recall the elementary fact from recursion theory that natural numbers may be used as codes for recursive functions. The letter e is traditionally used for such codes. We write $\{e\}$ for the partial function coded by e.

Let R and S be PERs. A partial function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is said to *track* (from R to S) if

$$mRn \Longrightarrow (fm)S(fn)$$

for all natural numbers m, n.

We define an equivalence relation $\sim$ between codes for such functions by

$$e \sim e' \quad \text{iff} \quad \forall n.\ nRn \Longrightarrow (\{e\}n)S(\{e'\}n)$$

for any codes e and e'. We write $[e]_{R,S}$ for the equivalence class of such a code (and often omit the subscripts). A map between PERs consists of just such an equivalence class of codes. Let *PER* be the category of partial equivalence relations and let PER_0 be its set of objects.

The category *PER* is cartesian closed. The terminal PER is such that every number is related to every other. Let

$$(-,-) : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

be one of the standard recursive operations for encoding pairs of natural numbers as natural numbers. Then the product $R \times S$ of PERs R and S is given by the condition

$$(m, n)(R \times S)(m', n') \quad \Longleftrightarrow \quad mRm' \ \&\ nSn'$$

for all $m, n, m', n' \in \mathbb{N}$ and their are corresponding first and second projections with codes p_1 and p_2, respectively. In fact *PER* has all finite limits. The exponent $R \to S$ is defined by

$$n(R \to S)n' \quad \text{iff} \quad \forall m, m'.\ mRm' \Longrightarrow (\{n\}m)S(\{n'\}m')$$

for any two pers R and S. The intersection of any (set-indexed) family of PERs is a PER.

The category *PER* 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 (and pairs of functions) is a CDCC (O'Hearn and Pym 1999; Pym 2002). Finite products and exponentials are given pointwise. Moreover, there is an additional symmetric monoidal closed structure with

$$\begin{aligned}(A^0, A^1) \otimes (B^0, B^1) &= ((A^0 \times B^0) + (A^1 \times B^1), (A^0 \times B^1) + (A^1 \times B^0)) \\ (A^0, A^1) \multimap (B^0, B^1) &= ((A^0 \to B^0) \times (A^1 \to B^1), (A^0 \to B^1) \times (A^1 \to B^0)) \ ,\end{aligned}$$

for all $A^0, A^1, B^0, B^1 \in \mathit{Set}$, where $A + B$ is the coproduct of A and B in Set. This can be viewed as an instance $\mathit{Set}^{\mathbf{2}}$ of Day's closure construction (Day 1970; Day 1973), where $\mathbf{2} = \{0, 1\}$ is the discrete category with monoid given by addition modulo two.

Definition 7. Define the category $\mathit{PER} \times \mathit{PER}$ to have objects consisting of pairs $R = (R^0, R^1)$ of PERs. Arrows of $\mathit{PER} \times \mathit{PER}$ are equivalence classes

$$[e]_{R,S} : R \longrightarrow S$$

of codes, where

— $R = (R^0, R^1)$ and $S = (S^0, S^1)$
— $e = (e^0, e^1) \in \mathbb{N}$, where $[e^0] : R^0 \longrightarrow S^0$ and $[e^1] : R^1 \longrightarrow S^1$
— the partial equivalence relation on such codes for pairs of codes is given by

$$e \sim f \iff e^0 \sim f^0 \ \text{ and } \ e^1 \sim f^1$$

for any two suitable codes for pairs of codes e, f.

As before, we usually omit the subscripts on equivalence classes. Note also that an arrow between objects is really just a coding for the evident pair of PER arrows.

The category $\mathit{PER} \times \mathit{PER}$ can be viewed as $\mathit{PER}^{\mathbf{2}}$ and so is known to be doubly closed, following (Day 1970; Day 1973). The product and exponential are lifted pointwise from PER. That is

$$\begin{aligned}R \times S &= (R^0 \times S^0, R^1 \times S^1) \\ R \to S &= (R^0 \to S^0, R^1 \to S^1)\end{aligned}$$

for all $R = (R^0, R^1)$ and $S = (S^0, S^1)$. The monoidal closed operations $(\otimes, \multimap)$ are defined in the same way as those of $\mathit{Set} \times \mathit{Set}$. Note 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 by taking

$$\mathit{dom}(A, B) = \mathit{dom}(A) \times \mathit{dom}(B)$$

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.

5.2. *A Hyperdoctrine*

Let X be a bunch of type variables. Let

$$\mathit{dom}(\rho) = \bigcup_{\alpha \in \mathit{ftv}(X)} \mathit{dom}(\rho(\alpha))$$

for any function $\rho : \mathit{ftv}(X) \longrightarrow \mathit{PER}_0 \times \mathit{PER}_0$.

Definition 8. An *environment* for X is a function $\rho : \mathit{ftv}(X) \longrightarrow \mathit{PER}_0 \times \mathit{PER}_0$ such that if any (Y, Z) is a sub-bunch of X then

$$\mathit{dom}(\rho\restriction_Y) \cap \mathit{dom}(\rho\restriction_Z) = \emptyset$$

holds. Let $\mathit{Env}(X)$ be the set of environments for X.

Definition 9. A *semantic type* (over X) is a function $\tau : \mathit{Env}(X) \longrightarrow \mathit{PER}_0 \times \mathit{PER}_0$ from environments to pairs of PERs.

A map from τ to τ' (over X) is an equivalence class $[e]$ of codes for pairs of codes, $e = (e^0, e^1)$, where the recursive function corresponding to each e^i is a PER map from $(\tau\rho)^i$ to $(\tau'\rho)^i$ for all environments ρ. If we have maps $[e] : \tau \longrightarrow \tau'$ and $[f] : \sigma \longrightarrow \tau$ then we have a composite map $[e] \circ [f] : \sigma \longrightarrow \tau'$ given by $[e] \circ [f] = [e.f]$ where $e.f = (e^0.f^0, e^1.f^1)$ and $e^i.f^i$ is a code for the recursive function defined by composite of the functions $\{e^i\}$ and $\{f^i\}$ for $i = 1, 2$. This gives a category $P(X)$ of semantic types over X.

These definitions give a natural generalization of a standard 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.

Definition 10. Let $\beta_1, \ldots, \beta_n$ be the variables of Y. A *(semantic) substitution*

$$s = [\tau_1/\beta_1, \ldots, \tau_n/\beta_n] : X \longrightarrow Y$$

for Y consists of a sequence of semantic types $\tau_1, \ldots, \tau_n$ over some bunch X. Furthermore, if β_i and β_j are multiplicatively combined in Y then

$$\mathit{dom}(\tau_i\rho) \cap \mathit{dom}(\tau_j\rho) = \emptyset$$

holds for every environment $\rho \in \mathit{Env}(X)$.

Note the special case of the nullary substitution, $[] : \emptyset \longrightarrow \emptyset$, when $n = 0$.

Lemma 13. Suppose τ is a semantic type over Y and $s = [\tau_1/\beta_1, \ldots, \tau_n/\beta_n] : X \longrightarrow Y$ is a substitution. Then there is a map

$$(-)^s : \mathit{Env}(X) \longrightarrow \mathit{Env}(Y)$$

given by

$$\rho^s(\beta_i) = \tau_i(\rho)$$

for every environment ρ for X. Moreover,

$$s^*(\tau) = \tau[\tau_1/\beta_1, \ldots, \tau_n/\beta_n]$$

is a semantic type over X with

$$s^*(\tau)(\rho) = \tau(\rho^s)$$

for every environment ρ of X.

The proof is immediate and uses the disjointness condition for substitution.

For each type variable α let τ_α be a semantic type defined over any hub X containing α by

$$\tau_\alpha(\rho) = \rho(\alpha)$$

for each environment ρ for X.

Let X have free variables $\alpha_1, \ldots \alpha_n$ for some $n \geq 0$. Then identity map from X to X is given by the substitution

$$[\tau_{\alpha_1}/\alpha_1, \ldots, \tau_{\alpha_n}/\alpha_n]$$

which may, of course, be nullary.

Given substitutions

$$s = [\sigma_1/\beta_1, \ldots, \sigma_n/\beta_n] : X \longrightarrow Y \qquad\qquad t = [\tau_1/\gamma_1, \ldots, \tau_p/\gamma_p] : Y \longrightarrow Z$$

we can form a composite substitution

$$t \circ s = [\tau_1'/\gamma_1, \ldots, \tau_p'/\gamma_p] : X \longrightarrow Z$$

where

$$\tau_i' = s^*(\tau_i)$$

for $1 \leq i \leq p$.

Definition 11. Let *Bun* be the category of hubs and semantic substitutions.

Let X be a hub with $ftv(X) = \{\alpha_1, \ldots, \alpha_n\}$. Then

$$[\tau_{\alpha_1}/\alpha_1, \ldots, \tau_{\alpha_n}/\alpha_n]$$

defines maps

$$\pi : X; \alpha \longrightarrow X \quad \text{ and } \quad \psi : X, \alpha \longrightarrow X$$

in *Bun*.

Lemma 14. The map π is a projection. The map ψ is a pseudoprojection.

Lemma 15. If s is the identity substitution on X then $\rho^s = \rho$ for every environment ρ for X. Given substitutions $s : X \longrightarrow Y$ and $t : Y \longrightarrow Z$ we have

$$\rho^{t \circ s} = (\rho^s)^t$$

for every environment ρ for X.

The proofs of both of the above lemmas are routine verifications.

Proposition 6. The assignments above define a functor $P : Bun^{op} \longrightarrow \mathbf{CDCC}$ (so is a strict indexed category) and this has a generic object.

Proof. Every category $P(X)$ is doubly closed with operations inherited pointwise from $PER \times PER$.

Suppose we are given a substitution $s = [\tau_1/\beta_1, \ldots, \tau_n/\beta_n] : X \longrightarrow Y$ in *Bun*. The substitution defines object and arrow assignments from $P(Y)$ to $P(X)$. Given any object

τ of $P(Y)$ we have $P(s)(\tau) = s^*(\tau)$ in $P(X)$. Given any map $f : \sigma \longrightarrow \tau$ in $P(Y)$ we define a map $P(s)(f) : P(s)(\sigma) \longrightarrow P(s)(\tau)$ in $P(X)$ by $P(s)(f)(\rho) = s^*(f)(\rho) = f(\rho^s)$ for any $\rho \in \mathit{ftv}(X)$. The fact that $P(s)$ is functorial follows almost immediately from the fact that composition is defined pointwise (with respect to environments) in the fibres. That P is functorial follows immediately from Lemma 15.

The generic object is given by (any choice of) a bunch consisting of a single type variable. Indeed, the arrows of the base category have been chosen so as to make this generic. □

Lemma 16. There are functors

$$P(\pi) : P(X) \longrightarrow P(X;\alpha) \qquad\qquad P(\psi) : P(X) \longrightarrow P(X,\alpha)$$

induced by projection $\pi : X;\alpha \longrightarrow X$ and pseudoprojection $\psi : X,\alpha \longrightarrow X$. For, respectively,

$$\rho \in \mathit{Env}(X;\alpha) \qquad\qquad \rho \in \mathit{Env}(X,\alpha)$$

we have

$$P(\pi)(\tau)(\rho) = \tau(\rho\restriction_{\mathit{ftv}(X)}) \qquad\qquad P(\psi)(\tau)(\rho) = \tau(\rho\restriction_{\mathit{ftv}(X)})$$

and

$$P(\pi)([e])(\rho) = [e] \qquad\qquad P(\psi)([e])(\rho) = [e]$$

for types τ and arrows $[e] : \tau \longrightarrow \tau'$ over X.

If ρ is an environment for X and $A \in \mathit{PER}_0 \times \mathit{PER}_0$ then define a function

$$\rho^A : \mathit{ftv}(X) \cup \{\alpha\} \longrightarrow \mathit{PER}_0 \times \mathit{PER}_0$$

by

$$\rho^A(\beta) = \begin{cases} A & \text{if } \beta = \alpha \\ \rho(\beta) & \text{if } \beta \neq \alpha \end{cases}$$

Now ρ^A is an environment for $X;\alpha$. If A satisfies $\mathit{dom}(A) \cap \mathit{dom}(\rho) = \emptyset$ then ρ^A is also an environment for X,α.

Lemma 17. There are functors

$$\Pi : P(X;\alpha) \longrightarrow P(X) \qquad\qquad \Psi : P(X,\alpha) \longrightarrow P(X)$$

between fibres.

The object assignments are, respectively,

$$(\Pi(\tau))(\rho) = \bigcap_{A \in \mathit{PER}_0 \times \mathit{PER}_0} \tau(\rho^A) \qquad\qquad (\Psi(\tau))(\rho) = \bigcap_{\substack{A \in \mathit{PER}_0 \times \mathit{PER}_0 \\ \mathit{dom}(A) \cap \mathit{dom}(\rho) = \emptyset}} \tau(\rho^A) \ ,$$

for each semantic type τ in the fibre that is the domain in each case and environment ρ for X.

The arrow assignments are, respectively,

$$\Pi([e]) = [e] \qquad\qquad \Psi([e]) = [e]$$

where $[e] : \tau \longrightarrow \tau'$ is an arrow in the domain.

These functors describe the additive and multiplicative polymorphic quantifiers.

Proposition 7. The functor Π is right-adjoint to $P(\pi)$. The functor Ψ is right-adjoint to $P(\psi)$.

Proof. 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 a code that defines an arrow from $P(\pi)(\tau)$ to τ'. In the second case, a map from τ to $\Psi(\tau')$ is a code that defines an arrow from $P(\psi)(\tau)$ to τ'. The disjointness condition in the indexing of the intersection defining $\Psi(\tau')$ over X matches precisely the disjointness conditions on environments for X, α. In fact, it easy to verify that

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

are natural bijections. □

Proposition 8. The Beck-Chevalley condition for additives holds.

We omit the proof, which consists of routine calculations.

Theorem 5. The functor $P : \mathit{Bun}^{op} \longrightarrow \mathbf{CDCC}$ is a model for additive polymorphism and first-order bunching.

To summarize, we have shown that there is the appropriate structure on *PER* to provide a model of additive polymorphism. Our claim that additive polymorphism is just ordinary polymorphism is supported by the interpretation in this model: it is essentially just the standard interpretation of a polymorphic quantifier. This claim is further reinforced by the way that additive quantification and weakening interact.

On the other-hand, there are counterexamples to the Beck-Chevalley condition for multiplicative quantification. For example, consider the empty PER $\mathbf{0}$ (empty relation) and total PER $\mathbf{1}$ (all pairs related). Let τ be the semantic type over the bunch $\top, \alpha$ that maps every environment to $\mathbf{0}$. Let ρ be the environment over the bunch $\top, \beta$ that maps β to $\mathbf{1}$. Consider the pseudoprojection $\psi : 1_{Bun} \otimes \beta \Longrightarrow 1_{Bun}$. We find that

$$(P(\psi) \circ \Psi)\tau\rho = \bigcap_{A \in PER_0 \times PER_0} \tau((\rho\restriction_\top)^A) = \bigcap_{A \in PER_0 \times PER_0} \mathbf{0} = \mathbf{0}$$

$$(\Psi \circ P(\psi))\tau\rho = \bigcap_{A \in \{PER_0 \times PER_0 \mid A \cap \rho(\beta) = \emptyset\}} \tau((\rho^A)\restriction_\alpha) = \bigcap_{A \in \emptyset} \mathbf{0} = \mathbf{1}$$

which are clearly not equal.

6. Existential Quantifiers

Existential quantifiers are available in the polymorphic lambda calculus $\lambda 2$ and are closely connected to the concept of abstract data type (Mitchell and Plotkin 1988). First-order additive and multiplicative existential quantifiers have been studied in (O'Hearn and Pym 1999; Pym 2002). Here we describe additive and multiplicative polymorphic existential quantification

Additive existential quantification, $\exists$, is quite straightforward to add to the system $\alpha 2\lambda 2$ but the multiplicative quantifier, $\exists_*$, is very delicate. In particular, it requires a number of side-conditions that interfere with the side-condition (†) used throughout $\alpha 2\lambda 2$. Rather than attempting to describe such a system in its full complexity, we first remove the universal quantifiers and instances of (†) before adding the existentials.

The grammars generating types and terms are extended as follows:

$$\begin{array}{lll} \tau & ::= & \ldots \mid \exists\alpha.\tau \mid \exists_*\alpha.\tau \\ M & ::= & \ldots \mid \langle\phi, M\rangle \mid \mathbf{unpack}\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \text{ in }\ M \\ & & \mid \langle X, \phi, M\rangle_* \mid \mathbf{unpack}_*\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \text{ in }\ M\ , \end{array}$$

where α and x are bound in **unpack** and $\mathbf{unpack}_*$ terms. Notice that terms may now contain bunches of type variables.

The notion of free type variable is extended so that

$$\mathit{ftv}(\exists\alpha.\tau) = \mathit{ftv}(\exists_*\alpha.\tau) = \mathit{ftv}(\tau) \setminus \{\alpha\}$$

$$\mathit{ftv}(\langle\phi, M\rangle : \exists\alpha.\tau) = (\mathit{ftv}(\phi) \cup \mathit{ftv}(M)) \setminus \{\alpha\}$$

$$\mathit{ftv}(\langle X, \phi, M\rangle : \exists_*\alpha.\tau) = (\mathit{ftv}(X) \cup \mathit{ftv}(\phi) \cup \mathit{ftv}(M)) \setminus \{\alpha\}$$

$$\begin{array}{rcl} \mathit{ftv}(\mathbf{unpack}\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \text{in}\ N) & = & \mathit{ftv}(M) \cup (\mathit{ftv}(N) \setminus \{\alpha\}) \\ & = & \mathit{ftv}(\mathbf{unpack}_*\ M\ \mathbf{as}\ \langle\alpha, x\rangle\ \text{in}\ N)\ . \end{array}$$

Just as with the multiplicative universal quantifier, we are forced to use an additional syntactic measure with the multiplicative existential. The set $\mathit{WR}(M)$ of *witnessing resources* of a term M is the set of type-variables that occur in the component X of any sub-term $\langle X, \phi, N\rangle_*$.

The rules for existentials, which follow the generalized forms for natural deduction introduced in (Prawitz 1978), are presented in Figure 5. Both of $(\exists E)$ and $(\exists_* E)$ are subject to the side-condition

$$\alpha \notin \mathit{ftv}(\Delta) \cup \mathit{ftv}(\sigma)$$

which is standard for the elimination of existentials. In addition, both are subject to the side-condition

$$\mathit{WR}(M) \cap \mathit{WR}(N) = \emptyset \qquad\qquad \alpha \notin \mathit{WR}(N)$$

because of the presence of the multiplicative. Furthermore, both rules $(\exists I)$ and $(\exists_* I)$ are subject to the condition $\mathit{ftv}(\tau) \subseteq \mathit{ftv}(X) \cup \{\alpha\}$. Finally, we require that $X, Z \vdash \Gamma$ for $(\exists_* I)$. The typed terms $(M : \tau)[\phi/\alpha]$ in the premises of $(\exists I)$ and $(\exists_* I)$ are formed by taking the raw (i.e. untyped term) M and replacing all occurrences of α by ϕ, and similarly in the type.

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.

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

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

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

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

Fig. 5. Existential rules

We must be a little bit careful with substitution because of the occurrences of bunches of witnessing resources in the terms. We use the notation $M[(Y, \psi)/\alpha]$ for the term formed by substituting Y for all witnessing occurrences of α and ψ for all other occurrences of α in M. We write $M[\psi/\alpha]$ when there are no witnessing occurrences of α in M.

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

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

and suitable ζ-conversions for existentials are also possible, provided no universal quantifiers are present. Notice how the hub 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 9. (Substitution)

— If $X \mid \Gamma \vdash M : \tau$ and $W \vdash \psi$ are derivable, $X[W/\beta]$ is well-formed, W does not contain any bound variables from Γ or τ, W is free from all witnessing resources in M and β is not among the witnessing resources of M then $X[W/\beta] \mid \Gamma[\psi/\beta] \vdash (M : \tau)[\psi/\beta]$ is also derivable.

— 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.

Both parts are proved by the standard kind of induction and we omit them for brevity. The only point worth noting is that the side-conditions on the existential rules have been chosen to match the conditions on these substitution rules so that we may permute all substitutions towards the leaves of derivations.

Proposition 10. (Subject-reduction) If $X \mid \Gamma \vdash M : \tau$ is derivable and there is a reduction $(X \mid M) \twoheadrightarrow (Y \mid N)$ then $Y \mid \Gamma \vdash N : \tau$ is derivable.

The Proposition follows more-or-less immediately from the substitution laws once one has correctly established the side-conditions on rules. The condition on the substitution law for terms 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.

The multiplicative existential does not have a simple interpretation in a hyperdoctrine and, in particular, we cannot just use a left-adjoint to the pseudoprojection substitution.

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 hub (see the substitution rule) and are only revealed by a subsequent use of the elimination rule $(\exists_* E)$, leading to a hub-changing β-conversion, as above.

There are two special cases of $(\exists_* I)$. The first is

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

together with the usual side-condition. In this case, the resources Y are completely hidden in the package $\langle Y, \phi, M\rangle$. This cannot be described as an adjoint in a hyperdoctrine in a straightforward way.

The second special case is

$$\frac{X, Y \mid \Gamma \vdash (M : \tau)[\phi/\alpha] \qquad Y \vdash \phi}{X, Y \mid \Gamma \vdash \langle Y, \phi, M\rangle : \exists_* \alpha.\tau} \tag{1}$$

with the usual conditions. Note, that none of the resources Y are hidden. This rule, together with $(\exists_* E)$, is precisely that which corresponds to existential quantification as left-adjoint to the substitution induced by a pseudoprojection.

Lemma 18. (Mate forms of existentials)

— The rules $(\exists I)$, $(\exists E)$ are equivalent to the pair of rules

$$\frac{X; \alpha \mid \Gamma(x : \tau) \vdash N : \sigma}{X \mid \Gamma(z : \exists\alpha.\tau) \vdash \textbf{unpack}\ \langle \alpha, x\rangle\ \textbf{as}\ z \text{ in } N : \sigma} \qquad \frac{X \mid \Gamma(z : \exists\alpha.\tau) \vdash N : \sigma}{X; \alpha \mid \Gamma(x : \tau) \vdash N[\langle \alpha, x\rangle/z] : \sigma}$$

together with appropriate side-conditions.

— The rule $(\exists_* E)$ together with the restricted rule (1) above are equivalent to the rules

$$\frac{X, \alpha \mid \Gamma(x : \tau) \vdash N : \sigma}{X \mid \Gamma(z : \exists_*\alpha.\tau) \vdash \textbf{unpack}_*\ \langle \alpha, x\rangle\ \textbf{as}\ z \text{ in } N : \sigma} \qquad \frac{X \mid \Gamma(z : \exists_*\alpha.\tau) \vdash N : \sigma}{X, \alpha \mid \Gamma(x : \tau) \vdash N[\langle \alpha, \alpha, x\rangle/z] : \sigma}$$

together with appropriate side-conditions.

This lemma is established by standard proof-theoretic drudgery using both parts of the substitution rules of Proposition 9.

To be a little more precise about the categorical structure required for existentials to be the expected adjoints in a $\alpha 2\lambda 2$-hyperdoctrine $P : \mathbb{B}^{op} \longrightarrow \mathbf{CDCC}$, we assume that there is a left-adjoint Σ to π^* arising from every projection $\pi : X \times \Omega \longrightarrow X$. Similarly, we assume that there is a left-adjoint Σ_* to every ψ^* arising from a pseudoprojection $\psi : X \otimes \Omega \longrightarrow X$. We also require Frobenius isomorphisms

$$A \times \Sigma B \xrightarrow{fr^{\times}} \Sigma((\pi^* A) \times B) \qquad\qquad A \otimes \Sigma B \xrightarrow{fr^{\otimes}} \Sigma((\pi^* A) \otimes B)$$

for the additive, and, whenever there is a multiplicative,

$$A \times \Sigma_* B \xrightarrow{fr^{\times}_*} \Sigma_*((\psi^* A) \times B) \qquad\qquad A \otimes \Sigma_* B \xrightarrow{fr^{\otimes}_*} \Sigma_*((\psi^* A) \otimes B)\ ,$$

where $A \in P(\llbracket X \rrbracket)$ and $B \in P(\llbracket X \rrbracket \times \Omega)$ in the first case and $B \in P(\llbracket X \rrbracket \otimes \Omega)$ in the second. The interpretations then work in essentially the standard way: in particular, combinations of $fr^{\times}$ and $fr^{\otimes}$ are used for the additive because of the depth of the existential type in the mate form of the rules for $\exists$; a similar comment applies to $fr^{\times}_*$ and $fr^{\otimes}_*$ for $\exists_*$.

7. Discussion

Polymorphic lambda-calculi have been the subject of a great deal of study since their introduction (Girard 1971; Girard 1972; Reynolds 1974), most often as extensions of an additive, first-order lambda-calculus. A number of authors have considered polymorphic extensions of linear type theories (Plotkin 1993; Bierman *et al.* 2000; Maneggia 2004; Birkedal *et al.* 2006). More recently, Biering, Birkedal and Torp-Smith have considered higher-order extensions of the bunched lambda calculus $\alpha\lambda$ (Biering *et al.* 2007). Further to this they have developed higher-order extensions of Separation Logic.

To our knowledge, all such calculi have used a form of polymorphism that is additive. The work presented in this paper diverges from those calculi because we consider multiplicative polymorphism. Of course, we also consider $\alpha\lambda$ extended with additive polymorphism — this was done independently of Biering, Birkedal and Torp-Smith. Furthermore, the two forms of polymorphism can be combined through the use of bunches of type variables. There are variant calculi with bunching at both, either or none of the two levels, all of which have sensible substitution and subject-reduction properties.

In order to get a version of the first-order substitution property (Proposition 1) in the presence of multiplicative polymorphism, we introduced a syntax for instantiation which is rather cumbersome. In particular, all type variables used in any application of the multiplicative quantifier are recorded within the term, and we place the side-condition (†) on the formation of certain terms in Figure 1. Thus the logic underlying the calculus is not particularly well-behaved. This appears to be a feature of multiplicative polymorphism rather than the bunched structure of contexts — the problem remains if we allow ordinary linear contexts, rather than bunches, and drop the additive quantifier. One way to eliminate the side-conditions would be to take a more restrictive, multiplicative version of substitution, in which the two arguments of the substitution are terms that must be typed over hubs that may be multiplicatively combined. Those rules of Figure 1 featuring (†) would be modified correspondingly by typing the premises over distinct hubs X and Y, and the conclusions over the multiplicative combination X, Y. However, this does not appear to sit well with our intention that the fibres over X should be CDCCs in any model (including the term model). Indeed, it is not even clear that the usual kind of indexed categorical model (hyperdoctrine) would be appropriate for such a calculus.

It has been shown above that partial equivalence relations provide a model of additive polymorphism together with first-order bunching. We have not shown that *PER* can support the structure required to model multiplicative polymorphism. Indeed, perhaps the fundamental outstanding problem regarding this work is to find a model that satisfies the multiplicative Beck-Chevalley condition of Definition 5. It seems that the technique we have used in §5 (ensuring environments of type variables map multiplicatively combined type variables to disjoint PERs) is incompatible with the multiplicative Beck-Chevalley condition. A comparsion with the way that $\alpha\lambda$-contexts are often interpreted may be helpful. The Day tensor construction (Day 1970; Day 1973) provides many useful doubly-closed categories. In particular, the functor category $Set^{\mathcal{F}}$, where $\mathcal{F}$ is the discrete category of finite sets is discussed in detail in (O'Hearn 2003). The category $\mathcal{F}$ carries a partial monoid that is given by union just when the arguments are disjoint sets, and that is undefined otherwise. An assignment is a mapping from variables to objects of $Set^{\mathcal{F}}$. Contexts are interpreted as maps from assignments to objects of $Set^{\mathcal{F}}$. In particular, for any assignment η and finite set S the set $[\![\Gamma, \Delta]\!]\eta S$ consists of pairs of elements $y \in [\![\Gamma]\!]\eta T$ and $z \in [\![\Delta]\!]\eta U$ where $S = T \cup U$ and $T \cap U = \emptyset$. The fact that (additive) weakening commutes with abstraction (for variables other than the variable abstracted) is related to the fact that the disjointness condition in this model regards the additional arguments (the finite sets) rather than the assignment. This is in contrast to the environments for hubs that we have used above, where the interpretation of the environment itself splits. As yet, we have been unable to adapt this method to multiplicative polymorphic abstraction.

Many types are known to be definable using polymorphic abstraction in $\lambda 2$. For example, the important class of computational types known as inductive types are definable from $\forall$ and $\rightarrow$: this includes the empty type, booleans, natural numbers, lists, and inductive (free) structures, see for example (Girard *et al.* 1989; Pierce 2002). Now consider definability in $\alpha 2\lambda 2$. The empty type, booleans, natural numbers, additive products and inductive types are all definable (from $\forall$ and $\rightarrow$) by immediate adaptations of the standard constructions. On the other-hand, it is unclear if $\vee$, $\exists$ or $*$ can be defined from $\forall$, $\rightarrow$, $\forall_*$ and $-\!\!*$. Let the depth of a node of a tree be defined in the usual way. Say that a leaf of a tree is deep if there are nodes of lesser depth, otherwise say that it is shallow. The elimination rules for $\vee$, $\exists$ and $*$ in $\alpha 2\lambda 2$ and $\alpha\lambda$ allow the type being eliminated to occur deep inside a context. In contrast, the standard encodings of $\vee$ and $\exists$ give rise to elimination laws in $\alpha 2\lambda 2$ in which the eliminated variable is shallow in the context.

An outline of some possible applications to calculi for memory-management has appeared (Collinson and Pym 2006). This calculus has primitives for the allocation and freeing of regions (sets of machine memory locations). It uses multiplicative polymorphic types to ensure that allocation always uses a fresh region (not overlapping other regions in use) and that a region must be disjoint from the others in use before it can this can happen. The calculus is an extension of a lambda-calculus presented for allocating and freeing individual memory locations (Berdine and O'Hearn 2006).

References

M. Abadi and G.D. Plotkin. (1990) A per model of polymorphism and recursive types. In *Logic in Computer Science, LICS '90*, pages 31–46. IEEE Press.

R. Atkey. (2004) A λ-Calculus for Resource Separation. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*, volume 3142 of *LNCS*, pages 158-170. Springer.

J. Berdine and P. O'Hearn. (2006) Strong update, disposal and encapsulation in bunched typing. In *Mathematical Foundations of Programming Semantics, MFPS '06*, volume 158 of *Electronic Notes in Theoretical Computer Science.* Elsevier.

B. Biering, L. Birkedal, and N. Torp-Smith. (2007) BI-hyperdoctrines, higher-order separation logic, and abstraction. To appear in *ACM Transactions on Programming Languages and Systems.* An earlier version appeared in Proceedings of European Symposium on Programming (ESOP'05) (2005) pages 233-247, Springer-Verlag.

G.M. Bierman, A.M. Pitts and C.V. Russo. (2000) Operational Properties of Lily, a Polymorphic Linear Lambda Calculus with Recursion. In *Proceedings, Fourth International Workshop on Higher Order Operational Techniques in Semantics (HOOTS 2000)*, volume 41 of *Electronic Notes in Theoretical Computer Science*, pages 70–88.

L. Birkedal, R.E. Møgelberg, and R.L. Petersen. (2006) Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 5(2):1–48.

L. Birkedal and H. Yang. (2007) Relational parametricity and separation logic. To appear in *Logical Methods in Computer Science.*

L. Cardelli and G. Longo. (1990) A semantic basis for quest. Technical Report 55, Systems Research Center, Digital Equipment Corporation.

M. Collinson and D. Pym. (2006) Bunching for regions and locations. In *Mathematical Foundations of Programming Semantics, MFPS '06*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 171–197. Elsevier.

M. Collinson, D. Pym, and E. Robinson. (2005) On Bunched Polymorphism (Extended Abstract). In *Computer Science Logic, CSL '05*, volume 3634 of *LNCS*, pages 36–50. Springer.

B.J. Day. (1970) On closed categories of functors. In *Proceedings of the Midwest Category Seminar*, volume 137 of *LNM.* Springer.

B.J. Day. (1973) An embedding theorem for closed categories. In *Proceedings of the Sydney Category Seminar 1972/73*, volume 420 of *LNM.* Springer.

M.J. Gabbay and A.M. Pitts. (2002) A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363.

J.-Y. Girard. (1971) Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupres dans l'analyse et la théorie des types. In J.E. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium*, pages 63–92. North-Holland.

J.-Y. Girard. (1972) *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII.

J.-Y. Girard. (1987) Linear logic. *Theoretical Computer Science*, 50(1):1–102.

J-Y. Girard, Y. Lafont, and P. Taylor. (1989) *Proofs and Types.* Cambridge University Press.

J.M.E. Hyland. (1988) A small complete category. *Annals of Pure and Applied Logic*, 40:135–165.

B. Jacobs. (1999) *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics.* Elsevier.

F.W. Lawvere. (1969) Adjointness in foundations. *Dialectica*, 23:281–296.

S. Mac Lane. (1971) *Categories for the Working Mathematician.* Springer. Second edition 1998.

S. Mac Lane and I. Moerdijk. (1992) *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Springer.

D.B. MacQueen, G. Plotkin, and R. Sethi. (1986) An ideal model for recursive polymorphic types. *Information and Control*, 71:95–130.

P. Maneggia (2004) *Models of linear polymorphism.* PhD. thesis, School of Computer Science, The University of Birmingham.

J.C. Mitchell and E. Moggi. (1991) Kripke style models for typed lambda calculus. *Ann. Pure and Appl. Logic*, 51(1):99–124.

J.C. Mitchell and G.D. Plotkin. (1988) Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10:470–502.

P. O'Hearn. (2003) On bunched typing. *Journal of Functional Programming*, 13:747–796.

P. O'Hearn and D. Pym. (1999) The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244.

F.J. Oles. (1982) *A Category-Theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, Syracuse, USA.

B.C. Pierce. (2002) *Types and Programming Languages.* M.I.T. Press.

G.D. Plotkin. (1993) Type theory and recursion. In *Logic in Computer Science, LICS '93*, page 374. IEEE Press.

D. Prawitz. (1978) Proofs and the meaning and completeness of logical constants. In J. Hintikka, I. Niiniluoto, and E. Saarinen, editors, *Essays on mathematical and philosophical logic*, pages 25–40. D. Reidel.

D. Pym, P. O'Hearn, and H. Yang. (2004) Possible worlds and resources: The semantics of **BI**. *Theoretical Computer Science*, 315(1):257–305.

D.J. Pym. (2002) *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series.* Kluwer Academic Publishers. Errata at: `http://www.cs.bath.ac.uk/~pym/BI-monograph-errata.pdf`.

J.C. Reynolds. (1974) Towards a theory of type structure. In *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, Berlin.

J.C. Reynolds. (1981) The essence of algol. In J.W. de Bakker and J.C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland.

J.C. Reynolds. (2002) Separation logic: a logic for shared mutable data structures. In *Logic in Computer Science, LICS '02*, pages 55–74. IEEE Press.

R.A.G. Seely. (1987) Categorical semantics for higher order polymorphic lambda calculus. *Journal of Symbolic Logic*, 52:969–989.

M. Tofte. (1998) A brief introduction to regions. In *International Symposium on Memory Management '88*, pages 186–195. ACM Press