1 Untyped Syntax

value types	A	::=	Unit	
		|	Bool	
		|	Case A	
		|	OSum	
		|	$A \times A$	
		|	$A * A$	
		|	$U\underline{B}$	
computation types	B	::=	$A \to \underline{B}$	
		|	$A \mathrel{-\!\!*} \underline{B}$	
		|	$F\ A$	
values	V	::=	tt	
		|	true	
		|	false	
		|	x	variable
		|	$\mathrm{inj}_V\ V$	injection into the open sum
		|	(V, V)	
		|	$\pi_1\ V$	
		|	$\pi_2\ V$	
		|	$(V * V)$	separating product
		|	$\rho_1\ V$	
		|	$\rho_2\ V$	
		|	thunk M	
computations	M	::=	$\mho$	need monad for this
		|	force V	
		|	ret V	
		|	$x \leftarrow M;\ N$	
		|	$M\ N$	
		|	$M@N$	
		|	$\lambda x\colon A.M$	
		|	$\alpha x\colon A.M$	separating function
		|	$\mathrm{newcase}_A\ x;\ M$	"allocate" a new case in the open sum
		|	match V with V $\{\mathrm{inj}x.M \mid N\}$	match on open sum
		|	if V then M else N	
value typing context	Γ	::=	$\varnothing$	
		|	$x : A$	
		|	$\Gamma; \Gamma'$	cartesian product
		|	$\Gamma \fatsemi \Gamma'$	monoidal product
stoup	Θ	::=	$\cdot$	
		|	$\bullet : B$	

stoup forms?

2 Typed Syntax

This version of the syntax is a combination of the ν-calculus [13] [12] of Pitts and Stark and the affine $\alpha\lambda$-calculus of O'Hearn [8] expressed in CBPV and with the addition of an extensible sum type. for (rule airity > 1) non separating connectives, contexts don't need to be appended(Γ; Δ)?

2.1 Structural Rules

; admits weakening, contraction, and exchange (cartesian closed structure)
⨟ admits weakening and exchange (semicartisian symmetric monoidal structure)

2.2 Value Typing

2.2.1 Unit

$$\frac{}{\Gamma \vdash_v \mathsf{tt} : \mathsf{Unit}}\ \text{Intro}$$

2.2.2 Bool

$$\frac{}{\Gamma \vdash_v \mathsf{true} : \mathsf{Bool}}\ \text{Intro}_1$$

$$\frac{}{\Gamma \vdash_v \mathsf{false} : \mathsf{Bool}}\ \text{Intro}_2$$

$$\frac{\Gamma \vdash_v V : \mathsf{Bool} \qquad \Gamma \mid \cdot \vdash_c M_1 : B \qquad \Gamma \mid \cdot \vdash_C M_2 : B}{\Gamma \mid \cdot \vdash_c \mathsf{if}\ V\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : B}\ \text{Elim}$$

2.2.3 Var

$$\frac{}{x : \mathrm{A} \vdash_v x : \mathrm{A}}\ \text{Intro}$$

2.2.4 OSum

$$\frac{\Gamma \vdash_v V_c : \mathsf{Case}\ \mathrm{A} \qquad \Gamma \vdash_v V : A}{\Gamma \vdash_v \mathsf{inj}_{V_c}\ V : \mathsf{OSum}}\ \text{Intro}$$

$$\frac{\Gamma \vdash_v V_T : \mathsf{Case}\ \mathrm{A} \qquad \Gamma \vdash_v V : \mathsf{OSum} \qquad \Gamma;\ (x : A) \mid \cdot \vdash_c M : B \qquad \Gamma \mid \cdot \vdash_c N : B}{\Gamma \mid \cdot \vdash_c \mathsf{match}\ V_T\ \mathsf{with}\ V\ \{\mathsf{inj}\ x.M \mid N\}}\ \text{Elim}$$

2.2.5 Value Product

$$\frac{\Gamma \vdash_v N : A_1 \qquad \Gamma \vdash_v M : A_2}{\Gamma \vdash_v (N, M) : A_1 \times A_2} \text{ Intro}$$

$$\frac{\Gamma \vdash_v P : A_1 \times A_2}{\Gamma \vdash_v \pi_1 P : A_1} \text{ Elim}_1$$

$$\frac{\Gamma \vdash_v P : A_1 \times A_2}{\Gamma \vdash_v \pi_2 P : A_2} \text{ Elim}_2$$

2.2.6 Separating Product

we have these because the monoidal product is semicartesian

$$\frac{\Gamma \vdash_v M : A_1 \qquad \Delta \vdash_v N : A_2}{\Gamma \fatsemi \Delta \vdash_v (M * N) : A_1 * A_2} \text{ Intro}$$

$$\frac{\Gamma \fatsemi \Delta \vdash_v P : A_1 * A_2}{\Gamma \vdash_v \rho_1 P : A_1} \text{ Elim}_1$$

$$\frac{\Gamma \fatsemi \Delta \vdash_v P : A_1 * A_2}{\Delta \vdash_v \rho_2 P : A_2} \text{ Elim}_2$$

2.2.7 Thunk

$$\frac{\Gamma \mid \cdot \vdash_c M : B}{\Gamma \vdash_v \text{thunk } M : UB} \text{ Intro}$$

$$\frac{\Gamma \vdash_v V : UB}{\Gamma \mid \cdot \vdash_c \text{force } V : B} \text{ Elim}$$

2.3 Computation Typing

2.3.1 Function

$$\frac{\Gamma;\, (x : A) \mid \cdot \vdash_c V : B}{\Gamma \mid \cdot \vdash_c (\lambda x : A.\, V) : A \to B} \text{ Intro}$$

$$\frac{\Gamma \mid \Theta \vdash_c M : A \to B \qquad \Delta \vdash_v N : A}{\Gamma;\, \Delta \mid \Theta \vdash_c MN : B} \text{ Elim}$$

2.3.2 Separating Function

$$\frac{\Gamma \fatsemi (x : A) \mid \cdot \vdash_c V : B}{\Gamma \mid \cdot \vdash_c (\alpha x : A.\, V) : A \mathrel{-\!\!*} B} \text{ Intro}$$

$$\frac{\Gamma \mid \Theta \vdash_c M : A \mathrel{-\!\!*} B \qquad \Delta \vdash_v N : A}{\Gamma \fatsemi \Delta \mid \Theta \vdash_c MN : B} \text{ Elim}$$

2.3.3 Return

$$\frac{\Gamma \vdash_v V : A}{\Gamma \mid \cdot \vdash_c \mathrm{ret}\ V : FA}\ \text{Intro}$$

$$\frac{\Gamma \mid \Theta \vdash_c M : FA \qquad \Gamma; (x : A) \mid \cdot \vdash_c N : B}{\Gamma \mid \Theta \vdash_c x \leftarrow M;\ N : B}\ \text{Elim}$$

2.3.4 Error

$$\frac{}{\Gamma \mid \cdot \vdash_c \mho : B}\ \text{Intro}$$

2.4 Equations

for effects (but also computation rules)

2.4.1 Allocation

The following are from Ian Stark's Categorial Models for Local Names [12]

1. swap - allocation swap
2. drop - (our monad might not support this)
3. Fresh - (a kind of congruence)

2.4.2 Dynamic Type

the prism laws here

1.

3 Semantics

3.1 Worlds

Normally, the value types in a CBPV would be interpreted as Sets. However, in our case this is insufficient as the interpretation of the extensible sum type, Osum, is dependent on what Cases have been "allocated" via newcase. Thus we interpret value types in a presheaf category on a category of worlds where objects are a map from a finite set to syntactic types. Let SynTy be a set of syntactic types that can be used in the OSum Case store.

$$\begin{array}{rcl} \mathrm{SynTy} & ::= & \mathrm{Unit} \\ & \mid & \mathrm{Bool} \\ & \mid & \mathrm{SynTy} \times \mathrm{SynTy} \end{array}$$

The denotation of these syntactic types is independent of what Cases have been allocated. Limiting the extensible sum type to contain only base types and products of base types allows us to temporarily avoid two separate issues.

- The first issue has to do with circularity of definitions. If we say a world is a mapping from a fininte set to semantic types and that semantic types are presheaves on the category of worlds, then we have tied ourselves in a knot. This issue is pointed out in, among other places, [15].

$$\text{world} \cong X \to \text{SemTy} \quad \text{SemTy} \cong [\mathbf{World}, \mathbf{Set}]$$

- The second issue is well-foundedness of the interpretation of OSum. Consider if we allow the extensible sum type to store the type (OSum → OSum) and try to recursively interpret OSum.

Here we define the category **World**[1] [2]. A world : $X \to SynTy$ is a map from a finite set into the set of syntactic types. Morphisms $f : X \to Y$ in **World** are injective functions $i : Y \hookrightarrow X$ such that the following triangle commutes. The finite sets here represent a finite collection of case symbols and the injective map i represents *renaming*. We may denote a world $X \xrightarrow{w}$ SynTy by w.

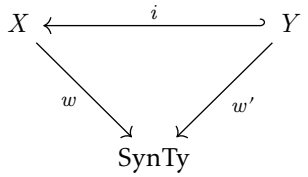


Figure 1: Morphisms in **World**

3.1.1 Monoidal Structure

We need to introduce additional structure on **World** to interpret the separating connectives $A * B$ and $A \mathrel{-\!\!*} B$. For this, we will pull from existing literature on categorical semantics of bunched typing, specifically [8] and section 3.1 of [10]. We use the Day convolution [2] to get a monoidal structure on $\widehat{\mathbf{World}}$[3] from a monoidal structure on **World**. First, we define an operation $w \otimes_w w'$ on *worlds* by taking the set coproduct of their domains[4] and using the recursor $X + Y \xrightarrow{rec\ w\ w'} SynTy$ to define a map into syntactic types. The action on

[1] Similar categories: $HeapV$ [11], $IFam_X$ [14], polymorphic algebraic theories [3], [16]

[2] Formalized here as **World** := $(Inc \downarrow \text{SynTy})^{op}$, where Inc is the inclusion functor from the category of fininte sets and injections into the category **Set** and SynTy is a constant functor.

[3] $\mathbf{Set}^{(\mathbf{World}^{op})}$; contravariant presheaves on **World**

[4] This is not the coproduct in FinSet_mono [1]

morphisms is given by defining the top injection via $map : (A \to C) \to (B \to D) \to A + B \to C + D$ and the commuting triangles still hold. The unit for this monoid structure is the empty map $\lambda()$ [5].

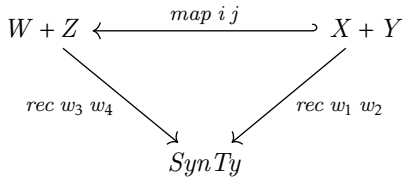


Figure 2: $\otimes_w$ action on morphisms

Using this operation, we define the monoidal product for $\widehat{\textbf{World}}$. The action on objects being:

$$(A \otimes B)X = \int^{Y,Z} A(Y) \times B(Z) \times \textbf{World}[X, Y \otimes_w Z]$$

The monoidal unit is given by:

$$I(X) = \textbf{World}[X, \lambda()]$$

The separating function is defined by:

$$(A \multimap B)X = \int_Z \textbf{Set}[A(Z), B(X \otimes_w Z)] \cong \widehat{\textbf{World}}[A, B(X \otimes_w \_)]$$

Finally, we have that:

$$\text{Hom}(A \otimes B, C) \cong \text{Hom}(A, B \multimap C)$$

These definitions are standard in the literature on bunched implication with the exception that our *resources* are finite maps specifying typing of allocated case symbols. The bicartesian stucture on $\widehat{\textbf{World}}$ along with the monoidal structure ($\otimes$) and its right adjoint ($\multimap$) bundled together gives us the structure of a bicartesian doubly closed category or BiDCC.

3.1.2 Affine BiDCC

The concrete model that we are working with has the property that $\mathbf{1} \cong I$ [6]. That is, the cartesian unit is isomorphic to the monoidal unit. This property validates weakening for $\S$. Note that:

$$\mathbf{1}(X) = \{*\}$$
$$I(X) = \textbf{World}[X, \lambda()]$$

$\lambda()$ is terminal in **World** and the cardinality of $|\,\textbf{World}[\_, \lambda()]\,| = 1$.

[5] The empty map is also the terminal object in **Worlds**
[6] See section 3.3 of [10]

3.1.3 Concrete Separating Connectives

It may be possible for us to have an alternative interpretation of the separating product if we work in a full subcategory of $\widehat{\mathbf{World}}$. O'Hearn has affine models [8] [7] for his $\alpha\lambda$-calculus [7] using a presheaf category on $\mathbf{FinSet}_{\text{mono}}$. He shows that(section 4 , [6]) the full subcategory of $\mathbf{Set}^{\mathbf{FinSet}_{\text{mono}}^{op}}$ which consists of pullback preserving functors lets you ask for the *smallest* world that any $a \in A(X)$ comes form, aka $support(a)$(Section 9.2, [7]). With this notion of support, you can define non-interference as:

$$a \# b \Leftrightarrow (support(a) \cap support(b) = \varnothing)$$

With non-interference, $A * B$ can be directly interpreted as:

$$(A * B)X = \{(a, b) \in A(X) \times B(X) \mid a \# b\}$$
$$(A * B)f(a, b) = (A(f)(a), B(f)(b))$$

It should be noted that the full subcategory mentioned above is equivalent to the Shanuel topos (Remark 6.9 [9]). It seems this would extend to our *multi-sorted* scenario[8] [16] [4].

3.2 Allocation Monad

To model allocation of case symbols, we can use a modified version of the indexed state monad [**?**] [**?**].

3.2.1 Limitations

There are plenty of limitations with our current approach to modeling the dynamic type. Leveraging the analogy with state [9], our model supports *dynamic* allocation of cases but only for *ground* types and allocation is *globally* observable. We can adapt Sterling's model [15] to store semantic types instead of syntactic types. Kammar has a candidate for local store [**?**]. It is unclear to me if local store with rich semantic types exists yet.

3.3 Interpreting Types

We will take $\widehat{\mathbf{World}}$ to be the value category in our model.

With all the categorical structure in place, we now give an interpretation of syntax. We first give an interpretation to syntactic types $SynTy$ via $[\![\_]\!]_{el} : SynTy \to \widehat{\mathbf{World}}$. We recall that the interpretation of these types is not dependent on what cases have been allocated. $\mathbf{1}$ is the terminal presheaf in $\widehat{\mathbf{World}}$

[7] And an extension with commands, memory cells, and assignment called he calls SCI+ which is based on Reynold's Syntactic Control of Interference

[8] The Schanuel topos models classical HOL, is this problematic?

[9] Using Jon's breakdown [14]

and $+, \times$ come from the bicartesian structure of $\widehat{\mathbf{World}}$.

$$\begin{aligned}
[\![Unit]\!]_{el} &= \mathbf{1} \\
[\![Bool]\!]_{el} &= \mathbf{1} + \mathbf{1} \\
[\![A \times B]\!]_{el} &= [\![A]\!]_{el} \times [\![B]\!]_{el}
\end{aligned}$$

Next, we interpret the remaining value types in $\widehat{\mathbf{World}}$. The separating product is interpreted as the monoidal product from the Day convolution and $|\, [\![B]\!]_c \,|$ is the forgetful functor which returns the carrier of the algebra $[\![B]\!]_c$. The main definitions of interest are the interpretation of $OSum$ and $Case$. $OSum$ is an extensible sum type where the current world determines the number and type of cases in the extensible sum. $Case\ A$ is interpreted to be the set of allocated case symbols for type A in the current world. Note that syntactic type equality is used in the set comprehension. This means that $\sigma : Case(A \times (B \times C))$ and $\sigma' : Case((A \times B) \times C)$ are distinct elements of two distinct sets. It may be nice to have a more flexible definition $\{\sigma \mid \sigma \in dom(\rho) \wedge [\![\rho(\sigma)]\!]_{el} \cong [\![A]\!]_{el}\}$. We would have to be more careful when constructing and destructing elements of $OSum$. Consider $\sigma : Case\ A$ and $\mathrm{inj}_\sigma(*, a) : OSum$. Should this be allowed with $[\![A]\!] \cong [\![Unit \times A]\!]$?

$$\begin{aligned}
[\![Unit]\!]_v &= [\![Unit]\!]_{el} \\
[\![Bool]\!]_v &= [\![Bool]\!]_{el} \\
[\![A \times B]\!]_v &= [\![A \times B]\!]_{el} \\
[\![OSum]\!]_v(\rho) &= \sum_{\sigma \in dom(\rho)} [\![\rho(\sigma)]\!]_{el}(\rho) \\
[\![Case\ A]\!]_v(\rho) &= \{\sigma \mid \sigma \in dom(\rho) \wedge \rho(\sigma) = A\} \\
[\![A * B]\!]_v &= [\![A]\!]_v \otimes [\![B]\!]_v \\
[\![U\, \underline{B}]\!]_v &= |\, [\![B]\!]_c \,|
\end{aligned}$$

Lastly, we interpret the computation types as their respective algebras in **TAlg**.

$$\begin{aligned}
[\![A \to B]\!]_c &= expAlg[\![A]\!]_v [\![B]\!]_c \\
[\![A \mathrel{-\!\!*} B]\!]_c &= sepAlg[\![A]\!]_v [\![B]\!]_c \\
[\![FA]\!]_c &= freeAlg[\![A]\!]_v
\end{aligned}$$

3.4 Interpreting Contexts

In *regular* type theory, contexts have the structure of a list and are interpreted as an n-ary product in the category used to interpret types. In the bunched type theory we are working with here, there are two connectives for constructing

contexts (";" and "⨟") and the structure of contexts are "tree-like". We use ; to denote the usual context construction operation that is interpreted as a cartesian product and ⨟ to denote the new context construction operation that is interpreted as the tensor product.

$$
\begin{aligned}
[\![x : A]\!]_{ctx} &= [\![A]\!]_v \\
[\![\phi]\!]_{ctx} &= \mathbf{1} \\
[\![\Gamma \,;\, \Gamma]\!]_{ctx} &= [\![\Gamma]\!]_{ctx} \times [\![\Gamma]\!]_{ctx} \\
[\![\varphi]\!]_{ctx} &= I \\
[\![\Gamma \fatsemi \Gamma]\!]_{ctx} &= [\![\Gamma]\!]_{ctx} \otimes [\![\Gamma]\!]_{ctx}
\end{aligned}
$$

The cartesian fragment of the context supports weakening and contraction. The monoidal fragment supports weakening[10].

3.5 Interpreting Terms

Value terms $\Gamma \vdash M : A$ are interpreted as morphisms $[\![\Gamma]\!]_{ctx} \xrightarrow{M} [\![A]\!]_v$ in $\widehat{\mathbf{World}}$ which are natural transformations from $\mathbf{World}^{\mathbf{op}}$ to **Set**. So for any term $\Gamma \vdash M : A$, we need to provide a family of maps $\forall\, w,\ \alpha_w : [\![\Gamma]\!](w) \to [\![A]\!](w)$ such that given any morphism $f : w \to w'$ in **World** we have:

$$
\begin{array}{ccc}
[\![\Gamma]\!](w) & \xleftarrow[[\![\Gamma]\!](f)]{} & [\![\Gamma]\!](w') \\
\downarrow{\scriptstyle \alpha_w} & & \downarrow{\scriptstyle \alpha_{w'}} \\
[\![A]\!](w) & \xleftarrow[[\![A]\!](f)]{} & [\![A]\!](w')
\end{array}
$$

Here we define the terms as natural transformations by giving the components. check: do we get naturality via parametric polymorphism in this setting? w is the current world and $\gamma : [\![\Gamma]\!](w)$. The base terms are straightforward.

$$
\begin{aligned}
[\![\mathsf{tt}]\!]_{vtm}(w, \gamma) &= * \\
[\![\mathsf{true}]\!]_{vtm}(w, \gamma) &= inl\ * \\
[\![\mathsf{false}]\!]_{vtm}(w, \gamma) &= inr\ *
\end{aligned}
$$

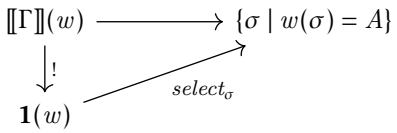
Deleted whole page, becase our monoidal product is affine, we have projections [5](Definition 2.1.ii) We can use this process (call it *lookup*) to construct

[10] consequence of $\mathbf{1} \cong I$ in our model

the proper denotation for variables.

$$[\![x]\!]_{vtm} = lookup(x)$$

For case symbols, we'd like a natural transformation $[\![\sigma]\!] : [\![\Gamma]\!] \to [\![Case\ A]\!]$ where components are defined as so:



which is definable as long as long as $\sigma \mapsto A \in w$. Consider the empty map $\lambda() \in Obj(\mathbf{Worlds})$, $\alpha_{\lambda()} : [\![\Gamma]\!](\lambda()) \to \varnothing$. This seems problematic..

$$[\![\sigma]\!]_{vtm}(w) = select_\sigma \circ !$$

For injection of values into the open sum type, we can assume we already have $[\![\Gamma]\!] \xrightarrow{\sigma} [\![Case\ A]\!]$ and $[\![\Gamma]\!] \xrightarrow{V} [\![A]\!]$. We then want to construct $[\![\text{inj}_\sigma\ V]\!] : [\![\Gamma]\!] \to [\![OSum]\!]$. First, lets consider the partially applied term $[\![\text{inj}_\sigma]\!] : [\![A]\!] \to [\![OSum]\!]$. Consider the subset $\{\alpha_w \mid w\ s.t.\ \sigma \mapsto A \in w\}$ of components of $[\![\text{inj}_\sigma]\!]$. These components will be of the form $\alpha_w = (inl \mid inr) \circ inr^*$ which will inject a value of type $[\![A]\!]$ into the open sum type at world w. $[\![\Gamma]\!] \xrightarrow{\sigma, V} [\![Case\ A]\!] \times [\![A]\!] \xrightarrow{\text{inj}_\sigma, id} [\![OSum]\!]^{[\![A]\!]} \times [\![A]\!] \xrightarrow{eval} [\![OSum]\!]$

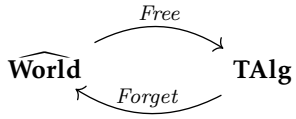
$$[\![\text{inj}_\sigma\ V]\!]_{vtm} = ?$$

The denotation of product and separating product are both akin to contructing bilinear maps.

$$[\![(V_1, V_2)]\!]_{vtm} = [\![V_1]\!], [\![V_2]\!]$$
$$[\![(V_1 * V_2)]\!]_{vtm} = [\![V_1]\!] * [\![V_2]\!]$$

And finally, the interpretation of thunk M is given by the forgetful functor which projects out the carrier of the algebra $[\![M]\!]_{ctm}$.

$$[\![\text{thunk}\ M]\!]_{vtm} = Forget([\![M]\!]_{ctm})$$



The interpretation of computation terms

$$
\begin{aligned}
[\![\mho]\!]_{ctm} &= \\
[\![\text{print } V]\!]_{ctm} &= \\
[\![\text{force } V]\!]_{ctm} &= [\![V]\!]_{vtm} \\
[\![\text{ret } V]\!]_{ctm} &= \mathit{Free}([\![V]\!]_{vtm}) \\
[\![x \leftarrow M;\ N]\!]_{ctm} &= \\
[\![M\ N]\!]_{ctm} &= \langle \mathit{eval}(|\,[\![M]\!]_{ctm}\,|)([\![N]\!]_{vtm}),\ \theta_{[\![M]\!]_{ctm}}[\![N]\!]_{vtm}\rangle \\
[\![M@N]\!]_{ctm} &= \langle \mathit{eval}^*(|\,[\![M]\!]_{ctm}\,|)([\![N]\!]_{vtm}),\ \theta_{[\![M]\!]_{ctm}}[\![N]\!]_{vtm}\rangle \\
[\![\lambda x : A.M]\!]_{ctm}(e : [\![\Gamma]\!]_v) &= \langle \lambda a : [\![A]\!]_v.\,|\,[\![M]\!]_{ctm}(e, a)\,|,\ \theta\rangle \\
[\![\alpha x : A.M]\!]_{ctm}(e : [\![\Gamma]\!]_v) &= \langle \alpha a : [\![A]\!]_v.\,|\,[\![M]\!]_{ctm}(e * a)\,|,\ \theta\rangle \\
[\![\text{newcase}_A\ \sigma;\ M]\!]_{ctm} &= \text{updated world and value environment?} \\
[\![\text{match } V_1 \text{ with } V_2\ \{x.M \mid N\}]\!]_{ctm} &= \\
[\![\text{let } (x, y) = V;\ M]\!]_{ctm} &= \\
[\![\text{let } (x * y) = V;\ M]\!]_{ctm} &= \\
[\![\text{if } V \text{ then } M \text{ else } N]\!]_{ctm} &=
\end{aligned}
$$

References

[1] https://math.stackexchange.com/questions/4797353/does-the-category-of-finite-sets-and-injective-functions-i-e-monomorphisms-h.

[2] https://ncatlab.org/nlab/show/day+convolutionday70thesis.

[3] Fiore, M., and Hamana, M. Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science* (New Orleans, LA, USA, June 2013), IEEE, pp. 520–529.

[4] Gabbay, M. J., and Pitts, A. M. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing 13*, 3-5 (July 2002), 341–363.

[5] Jacobs, B. Semantics of weakening and contraction. *Annals of Pure and Applied Logic 69*, 1 (1994), 73–106.

[6] O'Hearn, P. A Model for Syntactic Control of Interference. *College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects* (Jan. 1993).

[7] O'Hearn, P. On bunched typing. *Journal of Functional Programming 13*, 4 (July 2003), 747–796.

[8] O'Hearn, P. W. Resource Interpretations, Bunched Implications and the $A\lambda$-Calculus (Preliminary Version). In *Typed Lambda Calculi and Applications*, G. Goos, J. Hartmanis, J. Van Leeuwen, and J.-Y. Girard, Eds., vol. 1581. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 258–279.

[9] Pitts, A. M. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.

[10] Pym, D. J. *The Semantics and Proof Theory of the Logic of Bunched Implications*, vol. 26 of *Applied Logic Series*. Springer Netherlands, Dordrecht, 2002.

[11] Simpson, A. Category-theoretic structure for independence and conditional independence. *Electronic Notes in Theoretical Computer Science 336* (2018), 281–297. The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

[12] Stark, I. Categorical models for local names. *Lisp and Symbolic Computation 9*, 1 (Feb. 1996), 77–107.

[13] Stark, I. Names, equations, relations: Practical ways to reason about *new*. In *Typed Lambda Calculi and Applications: Proceedings of the Third International Conference TLCA '97* (1997), no. 1210 in Lecture Notes in Computer Science, Springer-Verlag, pp. 336–353. A full version appears as [**?**].

[14] Sterling, J., Gratzer, D., and Birkedal, L. Free theorems from univalent reference types: The impact of univalence on denotational semantics.

[15] Sterling, J., Gratzer, D., and Birkedal, L. Denotational semantics of general store and polymorphism, Apr. 2023. arXiv:2210.02169 [cs].

[16] Sterling, J., and Morrison, D. Syntax and Semantics of Abstract Binding Trees, Jan. 2016.