

1 Untyped Syntax

<i>value types</i>	A	$::=$	Unit Bool Case A OSum $A \times A$ $A * A$ $\underline{U \underline{B}}$	
<i>computation types</i>	B	$::=$	$A \rightarrow \underline{B}$ $A * \underline{B}$ $F A$	
<i>values</i>	V	$::=$	tt true false x $\text{inj}_V V$ (V, V) $\pi_1 V$ $\pi_2 V$ $(V * V)$ $\rho_1 V$ $\rho_2 V$ $\text{thunk } M$	<i>variable</i> <i>injection into the open sum</i>
<i>computations</i>	M	$::=$	$\bar{0}$ $\text{force } V$ $\text{ret } V$ $x \leftarrow M; N$ $M N$ $M @ N$ $\lambda x: A. M$ $\alpha x: A. M$ $\text{newcase}_A x; M$ $\text{match } V \text{ with } V \{ \text{inj } x. M \mid N \}$ $\text{if } V \text{ then } M \text{ else } N$	<i>separating product</i> <i>need monad for this</i>
<i>value typing context</i>	Γ	$::=$	\emptyset $x : A$ $\Gamma; \Gamma'$ $\Gamma \circ \Gamma'$	<i>separating function</i> <i>"allocate" a new case in the open sum</i> <i>match on open sum</i>
<i>stoup</i>	Θ	$::=$	\cdot $\bullet : B$	<i>cartesian product</i> <i>monoidal product</i>

stoup forms?

2 Typed Syntax

for (rule arity > 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 (semicartesian symmetric monoidal structure)

2.2 Value Typing

2.2.1 Unit

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

2.2.2 Bool

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

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

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

2.2.3 Var

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

2.2.4 OSum

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

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

2.2.5 Value Product

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

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

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

2.2.6 Separating Product

we have these because the monoidal product is semicartesian

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

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

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

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

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

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

2.3.2 Separating Function

$$\frac{\Gamma \circ (x : A) \mid \cdot \vdash_c V : B}{\Gamma \mid \cdot \vdash_c (\alpha x : A. V) : A \multimap B} \text{Intro}$$

$$\frac{\Gamma \mid \Theta \vdash_c M : A \multimap B \quad \Delta \vdash_v N : A}{\Gamma \circ \Delta \mid \Theta \vdash_c MN : B} \text{Elim}$$

2.3.3 Return

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

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

2.3.4 Error

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

2.4 Equations

for effects (but also computation rules) CITE The following are from Ian Stark's Categorical Models for Local Names

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

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.

$$\begin{array}{lcl} SynTy & ::= & Unit \\ & | & Bool \\ & | & SynTy \times SynTy \end{array}$$

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 finite 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, [?].

$$World \cong X \rightarrow_{fin} SemTy \quad SemTy \cong [Worlds, Set]$$

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

similar constructions: [Simpsons HeapV](#), [Stirling univalent reference \(or den\)](#), [Variable binding, nominal sets book](#)? Here we define the category **World**. A $world : X \rightarrow_{fin} SynTy$ is a finite map of syntactic types. Morphisms $f : X \rightarrow Y$ 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".¹

$$\begin{array}{ccc} X & \xleftarrow{i} & Y \\ & \searrow w & \swarrow w' \\ & SynTy & \end{array}$$

Previously, I had maps between worlds as "injections" where the source map was included in the domain map. I've flipped the ordering to make the DCC obviously affine(monoidal unit is isomorphic to cartesian unit).

3.1.1 Monoidal Structure

We need to introduce additional structure on $\widehat{\mathbf{Worlds}}$ to interpret the separating connectives $A * B$ and $A \multimap B$. For this, we will pull from existing literature on categorical semantics of bunched typing, specifically section 3.1 of [?]. In essence, we use the Day convolution [?] to get a monoidal structure on $\widehat{\mathbf{Worlds}}$ from a promonoidal structure on **Worlds**. First, we define an operation $x *_w y'$ on *worlds*.

$$x *_w y := \begin{cases} x \oplus y & \text{dom}(x) \cap \text{dom}(y) = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

where

$$(x \oplus y)(\sigma) := \begin{cases} x(\sigma) & \sigma \in \text{dom}(x) \\ y(\sigma) & \sigma \in \text{dom}(y) \end{cases}$$

The unit for this partial monoid structure is the empty map $\lambda()$ ². Using this operation³, we define the monoidal product for $\widehat{\mathbf{Worlds}}$:

$$(A \otimes B)X = \int^{Y,Z} A(Y) \times B(Z) \times \mathbf{Worlds}[X, Y *_w Z]$$

The concrete interpretation for this operation is

¹Formalized here as $\mathbf{World} := (Inc \downarrow SynTy)^{op}$, where Inc is the inclusion functor from the category of finite sets and injections into the category **Set** and $SynTy$ is a constant functor.

²The empty map is also the terminal object in **Worlds**

³ $*_w$ is only a partial operation on worlds, I'm skipping a step where the full promonoidal structure is defined from this

$$(A * B)X = \{[(Y, Z, a \in A(Y), b \in B(Z))] \mid Y *_w Z \text{ is defined and } X \leq Y *_w Z\}$$

where $[\cdot]$ denotes an equivalence class. $(Y, Z, a \in A(Y), b \in B(Z))$ and $(Y', Z', a' \in A(Y'), b' \in B(Z'))$ are equivalent if they have the same parent in the order determined by

$$\begin{aligned} (Y, Z, a \in A(Y), b \in B(Z)) &\leq (Y', Z', a' \in A(Y'), b' \in B(Z')) \\ \text{if} \\ f : Y &\leq Y', g : Z \leq Z', a' = A(f)(a), b' = B(g)(b) \end{aligned}$$

The monoidal unit is given by:

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

And the separating implication is defined by:

$$(A \multimap B)X = \int_Z \mathbf{Set}[A(Z), B(X *_w Z)] \cong \widehat{\mathbf{Worlds}}[A, B(X *_w -)]$$

Additionally we have that the tensor is left adjoint to magic wand $Hom(A \otimes B, C) \cong Hom(A, B \multimap C)$. These definitions are standard in the literature on bunched implication with the exception that we specify that our "resources" are a specific kind of finite maps. The bicartesian structure on $\widehat{\mathbf{Worlds}}$ along with the monoidal structure and its right adjoint bundled together has the structure of a bicartesian doubly closed category or BiDCC⁴.

3.1.2 Affine BiDCC

The concrete model that we are working with has the property that $\mathbf{1} \cong I$ ⁵. That is, the cartesian unit is isomorphic to the monoidal unit. This property validates weakening for \circ . Note that

$$\begin{aligned} \mathbf{1}(X) &= \{*\} \\ I(X) &= \mathbf{Worlds}[X, \lambda()] \end{aligned}$$

$\lambda()$ is terminal in \mathbf{Worlds} and the cardinality of $|\mathbf{Worlds}[_-, \lambda()]| = 1$.

3.2 Effect Monad

We need an appropriate monad on $\widehat{\mathbf{Worlds}}$ to interpret printing and error effects. First, we define an endofunctor T on $\widehat{\mathbf{Worlds}}$.

$$T_0(X) = \mathit{Const}_E + (X \times \mathit{Const}_{\mathit{FooBar}})$$

⁴Terrible name, but I'm just following the literature here

⁵TODO, work this out in full detail

⁶See section 3.3 of [?]

where $FooBar$ is a set containing all possible sequences of symbols "foo" and "bar" ($FooBar := \{s \mid s = (foo \mid bar)^+\}$) and $Const_{FooBar}$ is the constant presheaf mapping all *worlds* to set $FooBar$. E is a singleton set representing the "error state".

The action on morphisms is define as:

$$T_1(f : X \rightarrow Y)(Z) := \begin{cases} Z & Z : E \\ (f(x), s) & Z = (x, s) \end{cases}$$

where f is a natural transformation between presheaves (overloading notation here).

- functor laws
- natural transformations (return and join)
- monad laws
- Show T is strong (define strength nt and show 4 diagrams commute <https://ncatlab.org/nlab/show/strong+monad>) **strength for \otimes**

3.3 Computation Semantics

To interpret computation types, we use a free and forgetful adjunction between $\widehat{\mathbf{Worlds}}$ and the Eilenberg Moore category of monad T denoted \mathbf{TAlg} . \mathbf{TAlg} has all limits which exist in \mathbf{Worlds} ⁷. Concrete constructions can be found here [?]. The singleton, exponential, product, and free algebra are "given", what remains is to construct the separating implication algebra. **Assuming**⁸ T is a strong monad on \otimes , we have a map $T_{A \otimes B} : A \otimes T(B) \rightarrow T(A \otimes B)$, then the construction of the exponent algebra mirrors that of the separating algebra. I will reproduce the construction here.

Given an object $A : \widehat{\mathbf{Worlds}}$ and algebra $\langle B, \theta_B : T(B) \rightarrow B \rangle$, we need to construct an algebra. Set the carrier of the algebra to be $A \ast B$, we then need a map $\theta_{A \ast B} : T(A \ast B) \rightarrow (A \ast B)$. Via the tensor wand adjunction, it suffices to construct a map $T(A \ast B) \otimes A \rightarrow B$.

$$\begin{array}{ccc} T(A \ast B) \otimes A & \xrightarrow{\theta_{A \ast B}} & B \\ \downarrow T_{str} & & \uparrow \theta_B \\ T(A \otimes (A \ast B)) & \xrightarrow{T_1(eval_*)} & T(B) \end{array}$$

What remains is to check the T -algebra laws. We call $\langle A \ast B, \theta_{A \ast B} \rangle$ the separating algebra.

⁷<https://ncatlab.org/nlab/show/Eilenberg-Moore+category>

⁸TODO: show

3.4 Interpreting Types

With all the categorical structure in place, we now give an interpretation of syntax. We first give an interpretation to syntactic types $SynTy$ via $\llbracket - \rrbracket_{el} : SynTy \rightarrow \widehat{\mathbf{Worlds}}$. We recall that the interpretation of these types is not dependent on what cases have been allocated. $\mathbf{1}$ is the terminal presheaf in $\widehat{\mathbf{Worlds}}$ and $+$, \times come from the bicartesian structure of $\widehat{\mathbf{Worlds}}$.

$$\begin{aligned}\llbracket Unit \rrbracket_{el} &= \mathbf{1} \\ \llbracket Bool \rrbracket_{el} &= \mathbf{1} + \mathbf{1} \\ \llbracket A \times B \rrbracket_{el} &= \llbracket A \rrbracket_{el} \times \llbracket B \rrbracket_{el}\end{aligned}$$

Next, we interpret the remaining value types in $\widehat{\mathbf{Worlds}}$. The separating product is interpreted as the monoidal product from the Day convolution and $\llbracket B \rrbracket_c$ is the forgetful functor which returns the carrier of the algebra $\llbracket B \rrbracket_c$. 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 $\{\sigma \mid \sigma \in dom(\rho) \wedge \llbracket \rho(\sigma) \rrbracket_{el} \cong \llbracket A \rrbracket_{el}\}$. We would have to be more careful when constructing and destructing elements of $OSum$. Consider $\sigma : Case A$ and $inj_\sigma(*, a) : OSum$. Should this be allowed with $\llbracket A \rrbracket \cong \llbracket Unit \times A \rrbracket$?

$$\begin{aligned}\llbracket Unit \rrbracket_v &= \llbracket Unit \rrbracket_{el} \\ \llbracket Bool \rrbracket_v &= \llbracket Bool \rrbracket_{el} \\ \llbracket A \times B \rrbracket_v &= \llbracket A \times B \rrbracket_{el} \\ \llbracket OSum \rrbracket_v(\rho) &= \sum_{\sigma \in dom(\rho)} \llbracket \rho(\sigma) \rrbracket_{el}(\rho) \\ \llbracket Case A \rrbracket_v(\rho) &= \{\sigma \mid \sigma \in dom(\rho) \wedge \rho(\sigma) = A\} \\ \llbracket A * B \rrbracket_v &= \llbracket A \rrbracket_v \otimes \llbracket B \rrbracket_v \\ \llbracket U \underline{B} \rrbracket_v &= \llbracket B \rrbracket_c\end{aligned}$$

Lastly, we interpret the computation types as their respective algebras in \mathbf{TAlg} .

$$\begin{aligned}\llbracket A \rightarrow B \rrbracket_c &= expAlg \llbracket A \rrbracket_v \llbracket B \rrbracket_c \\ \llbracket A \multimap B \rrbracket_c &= sepAlg \llbracket A \rrbracket_v \llbracket B \rrbracket_c \\ \llbracket FA \rrbracket_c &= freeAlg \llbracket A \rrbracket_v\end{aligned}$$

3.5 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.

$$\begin{aligned} \llbracket x : A \rrbracket_{ctx} &= \llbracket A \rrbracket_v \\ \llbracket \phi \rrbracket_{ctx} &= \mathbf{1} \\ \llbracket \Gamma ; \Gamma \rrbracket_{ctx} &= \llbracket \Gamma \rrbracket_{ctx} \times \llbracket \Gamma \rrbracket_{ctx} \\ \llbracket \varphi \rrbracket_{ctx} &= I \\ \llbracket \Gamma \text{ ⋈ } \Gamma \rrbracket_{ctx} &= \llbracket \Gamma \rrbracket_{ctx} \otimes \llbracket \Gamma \rrbracket_{ctx} \end{aligned}$$

The cartesian fragment of the context supports weakening and contraction. The monoidal fragment supports weakening⁹.

3.6 Interpreting Terms

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

$$\begin{array}{ccc} \llbracket \Gamma \rrbracket(w) & \xleftarrow{\llbracket \Gamma \rrbracket(f)} & \llbracket \Gamma \rrbracket(w') \\ \downarrow \alpha_w & & \downarrow \alpha_{w'} \\ \llbracket A \rrbracket(w) & \xleftarrow{\llbracket A \rrbracket(f)} & \llbracket A \rrbracket(w') \end{array}$$

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 $\gamma : \llbracket \Gamma \rrbracket(w)$. The base terms are straightforward.

$$\begin{aligned} \llbracket \text{tt} \rrbracket_{vtn}(w, \gamma) &= * \\ \llbracket \text{true} \rrbracket_{vtn}(w, \gamma) &= \text{inl } * \\ \llbracket \text{false} \rrbracket_{vtn}(w, \gamma) &= \text{inr } * \end{aligned}$$

⁹consequence of $\mathbf{1} \cong I$ in our model

Deleted whole page, because our monoidal product is affine, we have projections [?](Definition 2.1.ii) We can use this process (call it *lookup*) to construct the proper denotation for variables.

$$\llbracket x \rrbracket_{vtn} = \text{lookup}(x)$$

For case symbols, we'd like a natural transformation $\llbracket \sigma \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \text{Case } A \rrbracket$ where components are defined as so:

$$\begin{array}{ccc} \llbracket \Gamma \rrbracket(w) & \xrightarrow{\quad} & \{\sigma \mid w(\sigma) = A\} \\ \downarrow ! & \nearrow \text{select}_\sigma & \\ \mathbf{1}(w) & & \end{array}$$

which is definable as long as long as $\sigma \mapsto A \in w$. Consider the empty map $\lambda() \in \text{Obj}(\mathbf{Worlds})$, $\alpha_{\lambda()} : \llbracket \Gamma \rrbracket(\lambda()) \rightarrow \emptyset$. This seems problematic..

$$\llbracket \sigma \rrbracket_{vtn}(w) = \text{select}_\sigma \circ !$$

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

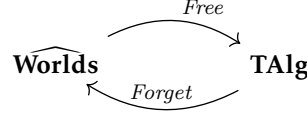
$$\llbracket \text{inj}_\sigma V \rrbracket_{vtn} = ?$$

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

$$\begin{aligned} \llbracket (V_1, V_2) \rrbracket_{vtn} &= \llbracket V_1 \rrbracket, \llbracket V_2 \rrbracket \\ \llbracket (V_1 * V_2) \rrbracket_{vtn} &= \llbracket V_1 \rrbracket * \llbracket V_2 \rrbracket \end{aligned}$$

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

$$\llbracket \text{thunk } M \rrbracket_{vtn} = \text{Forget}(\llbracket M \rrbracket_{ctm})$$



The interpretation of computation terms

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

A

References

- [1] <https://ncatlab.org/nlab/show/day+convolutionday70thesis>.
- [2] FORSTER, Y., SCHÄFER, S., SPIES, S., AND STARK, K. Call-by-push-value in Coq: operational, equational, and denotational theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Cascais Portugal, Jan. 2019), ACM, pp. 118–131.
- [3] JACOBS, B. Semantics of weakening and contraction. *Annals of Pure and Applied Logic* 69, 1 (1994), 73–106.
- [4] PYM, D. J. *The Semantics and Proof Theory of the Logic of Bunched Implications*, vol. 26 of *Applied Logic Series*. Springer Netherlands, Dordrecht, 2002.
- [5] STERLING, J., GRATZER, D., AND BIRKEDAL, L. Denotational semantics of general store and polymorphism, Apr. 2023. arXiv:2210.02169 [cs].