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
		|	$\text{inj}_V\ V$	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
		|	$\text{newcase}_A\ x;\ M$	"allocate" a new case in the open sum
		|	match V with V $\{\text{inj}x.M \mid N\}$	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? could add computation products

2 Typed Syntax

This version of the syntax is a combination of the ν-calculus [**?**] [**?**] of Pitts and Stark and the affine $\alpha\lambda$-calculus of O'Hearn [**?**] 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

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

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

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

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

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

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

2.3.4 New Case

$$\frac{\Gamma \fatsemi (x : CaseA) \mid \Theta \vdash_c M : B}{\Gamma \mid \Theta \vdash_c \mathsf{newcase}_A\ x;\ M}\ \text{new}$$

2.3.5 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 [**?**]

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 https://hackage.haskell.org/package/lens-5.3.2/docs/Control-Lens-Prism.htmlhere

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.

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

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, [**?**].

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

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

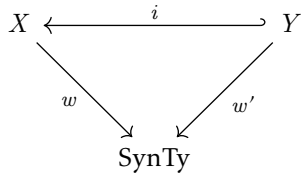

Figure 1: Morphisms in **World**

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 [**?**] and section 3.1 of [**?**]. We use the Day convolution [**?**] to get a monoidal structure on $\widehat{\mathbf{World}}$[3]

[1] Similar categories: $HeapV$ [**?**], $IFam_X$ [**?**], polymorphic algebraic theories [**?**], [**?**]

[2] Formalized https://github.com/bond15/Bunched-CBPV/blob/68f8b4d006edc9df1d830d7f9cc63822ec77a379/src/Data/Worlds.ag 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**

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 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].

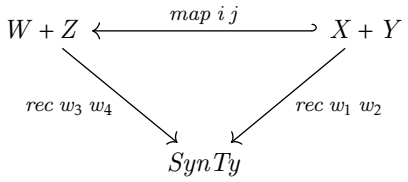

Figure 2: $\otimes_w$ action on morphisms

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:

[4] This is not the coproduct in FinSet_mono [**?**]
[5] The empty map is also the terminal object in **Worlds**
[6] See section 3.3 of [**?**]

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

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

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 [**?**] [**?**] for his $\alpha\lambda$-calculus [7] using a presheaf category on $\mathbf{FinSet}_{\text{mono}}$. He shows that(section 4 , [**?**]) 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, [**?**]). 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 [**?**]). It seems this would extend to our *multi-sorted* scenario[8] [**?**] [**?**]. See Biering's masters thesis [**?**](section 4.4) for a note on when the day tensor does have a right adjoint in the subcategory of sheaves.

3.2 Allocation Monad

To model allocation of case symbols, we can use a modified version of the indexed state monad [**?**] [**?**]. This monad is a special case of a general construction[9]. Given a functor $F : C \to D$ between two categories, it induces a functor on presheaf categories $F^* : [D^{op}, Set] \to [C^{op}, Set]$ via precomposition of F which has both a left and right adjoint given by Kan extensions.

We instantiate this construction twice [10] , using the inclusion functor from the discrete category $Inc : |\mathbf{World}| \to \mathbf{World}$ and its opposite Inc^{op}. This yields a monad on our value category, contravariant presheaves on **World**, by composing the top maps with the bottom maps[11].

[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] https://ncatlab.org/nlab/show/geometric+morphismBetweenPresheafToposes

[10] code https://github.com/bond15/Bunched-CBPV/blob/899a80968c055f086069b409b1f85ffb5a9d9aa5/src/Models/FuturePast.agd

[11] The usual presentation of the indexed state monad has covariant presheaves for the value category. ours is flipped to accomodate the definition of **World**

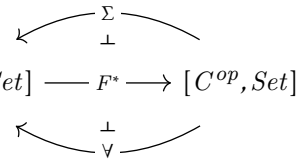

Figure 3: essential geometric morphism between presheaf topoi

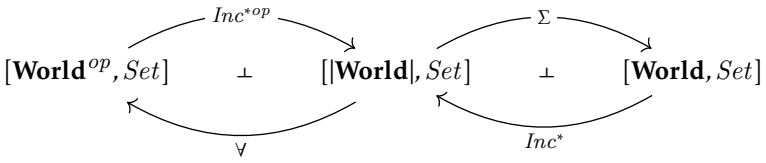

Figure 4: our allocation monad

The concrete interpretation for this monad's action on some object X at some world w_1 is: forall future worlds w_2 reachable by w_1, there exists some yet future world w_3 reachable from w_2 with a value $X(w_3)$.

$$T(X)(w_1) = \forall_{w_2}\, \forall_{w_1 \to w_2}\, \Sigma_{w_3} \Sigma_{w_2 \to w_3} X(w_3)$$

3.2.1 Limitations

There are plenty of limitations with our current approach to modeling the dynamic type. Leveraging the analogy with state [12], our model supports *dynamic* allocation of cases but only for *ground* types and allocation is *globally* observable. We can adapt Sterling's model [**?**] to store semantic types instead of syntactic types. Kammar has a candidate monad for local store [**?**]. It is unclear to me if local store with rich semantic types exists yet.

3.3 Interpreting Types

Taking contravariant presheaves on **World** as our value category and covariant presheaves on **World** as our computation category, we can give a denotation of the syntax. Interpretation of syntactic types $[\![\_]\!]_{el} : \text{SynTy} \to \widehat{\mathbf{World}}$ is not dependent on what cases have been allocated. $\mathbf{1}$ is the terminal presheaf in $\widehat{\mathbf{World}}$ and $+, \times$ come from the bicartesian structure of $\widehat{\mathbf{World}}$.

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

[12] Using Jon's breakdown [**?**]

Next, we interpret the remaining value types in $\widehat{\mathbf{World}}$. 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 [13] is used in the set comprehension.

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

Lastly, we interpret the computation types[14]

$$
\begin{aligned}
[\![A \to B]\!]_c(w) &= \mathbf{Set}[[\![A]\!]_v(w), [\![B]\!]_c(w)] \\
[\![A \multimap B]\!]_c(w) &= \int_{w'} \mathbf{Set}[[\![A]\!]_v(w'), [\![B]\!]_c(w \otimes_w w')] \\
[\![FA]\!]_c &= F [\![A]\!]_v
\end{aligned}
$$

[13] Although, I currently use the https://github.com/bond15/Bunched-CBPV/blob/1538ee3b3e1da806ec44bf635b35a5284ebb4924/src/Models/FuturePast.agdaL201usual path type in the Agda development. We might want Sterling's univalent reference types [**?**]

[14] The denotation for the regular function type is somewhat surprising. See section 7.1 of Levy's thesis (or 6.7.3 of the CBPV book). Levy's model for cell generation has:

$$[\![A \to B]\!]_c(w) = \mathbf{Set}[[\![A]\!]_v(w), [\![B]\!]_c(w)]$$

That is, the denotation of the computational function type is given pointwise where A is a covariant presheaf, and B is a contravariant presheaf and the result should be a contravariant presheaf(stuff is oped in our case). This functor does work in our situation. However, from O'Hearn's affine $\alpha\lambda$-calculus, we would expect this to also work:

$$[\![A \to B]\!](w) = \mathbf{Set}^{\mathbf{World}}[[\![A]\!]_v(w \otimes_w \_), [\![B]\!]_c(w \otimes_w \_)]$$

The situation is now more complicated because those monoidal operations exist in two different categories! (need to check this!) Additionally, this doesn't type check since the homset is taking objects from different categories. For the separating function, we should expect

$$[\![A \multimap B]\!](w) = \mathbf{Set}^{\mathbf{World}}[[\![A]\!]_v, [\![B]\!]_c(w \otimes_w \_)]$$

We'll probably have to use the set based end formula instead to bring another world into scope and fix the typing issues.

$$[\![A \multimap B]\!](w) = \int_{w'} \mathbf{Set}[[\![A]\!]_v(w'), [\![B]\!]_c(w \otimes_w w')]$$

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.

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

The cartesian fragment of the context supports weakening and contraction. The monoidal fragment supports weakening [15].

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 $\mathbf{World}^{\mathbf{op}}$ we have:

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

3.5.1 Value Terms

Here we define the terms as natural transformations by giving the components. w is the current world and $\gamma : [\![\Gamma]\!](w)$.

$$[\![\mathsf{tt}]\!]_{vtm}(w, \gamma) = *$$

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

$$[\![\text{true}]\!]_{vtm}(w,\gamma) = inl\ *$$
$$[\![\text{false}]\!]_{vtm}(w,\gamma) = inr\ *$$

Extracting a variable from the context is a matter of using the correct cartesian and monoidal projections [16]. We can use this process (call it *lookup*) to construct the proper denotation for variables.

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

For injection of values into the open sum type, we can assume we already have $[\![\Gamma]\!] \xrightarrow{\sigma} [\![\text{Case } A]\!]$ and $[\![\Gamma]\!] \xrightarrow{V} [\![A]\!]$. We then want to construct $[\![\text{inj}_\sigma\, V]\!] : [\![\Gamma]\!] \to [\![\text{OSum}]\!]$. It suffices to construct $[\![\text{Case} A \times A]\!] \to [\![\text{OSum}]\!]$ [17].

$$[\![\text{inj}_\sigma\, V]\!]_{vtm}(w,((x,p : w(x) = A),a)) = (x,a)$$

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

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

Projection maps are just postcomposing with the correct projections

$$[\![\pi_1\, V]\!] = [\![\Gamma]\!]_v \xrightarrow{[\![V]\!]_v} [\![A_1 \times A_2]\!] \xrightarrow{\pi_1} [\![A_1]\!]$$
$$[\![\pi_2\, V]\!] = [\![\Gamma]\!]_v \xrightarrow{[\![V]\!]_v} [\![A_1 \times A_2]\!] \xrightarrow{\pi_2} [\![A_2]\!]$$
$$[\![\rho_1\, V]\!] = [\![\Gamma \fatsemi \Delta]\!]_v \xrightarrow{[\![V]\!]_v} [\![A_1 * A_2]\!] \xrightarrow{\rho_1} [\![A_1]\!]$$
$$[\![\rho_1\, V]\!] = [\![\Gamma \fatsemi \Delta]\!]_v \xrightarrow{[\![V]\!]_v} [\![A_1 * A_2]\!] \xrightarrow{\rho_2} [\![A_2]\!]$$

Before we give the interpretation of thunk M, we need to discuss the interpretation of computation judgments.

[16] monoidal projections given by a bilinear map $! \otimes id$ composed with the monoidal identity laws [**?**](Definition 2.1.ii)

[17] It is morally transport p a in the https://github.com/bond15/Bunched-CBPV/blob/a2da10ec10f7bedcce8ded4aea6646b3a184d0b4/src/Models/FuturePast.agdaL264 formalization

3.5.2 Computation Terms

Following Levy's model [18] , the denotation of computation judgments are not morphisms in the computation category. Instead, they are denoted as a family of set maps:

$$[\![\Gamma \mid \Theta \vdash_c M : B]\!] = \forall_w \mathbf{Set}[[\![\Gamma]\!](w), [\![B]\!](w)]$$

I'm not satisfied with this. One alternative that typechecks is $\mathbf{Set}^{\mathbf{World}}[F[\![\Gamma]\!]_v, [\![B]\!]_c]$, but this is unnatural.

And finally, the interpretation of thunk M given $[\![M]\!](w) = [\![\Gamma]\!](w) \to [\![B]\!](w)$.

$$\begin{aligned}
&[\![\text{thunk } M]\!]_{vtm}(w)(\gamma : [\![\Gamma]\!](w)) = \lambda(w')(f : w' \to w).[\![M]\!](w)([\![\Gamma]\!]_1(f)(\gamma)) \\
&[\![\text{force } V]\!]_{ctm}(w)(\gamma : [\![\Gamma]\!](w)) = [\![V]\!]_{vtm}(w)(\gamma)(w)(id_w) \\
&[\![\text{ret } v]\!]_{ctm}(w)(\gamma : [\![\Gamma]\!](w)) = (w, id_w, [\![V]\!]_{vtm}(w)(\gamma)) \\
&[\![\lambda x : A.M]\!]_{ctm}(w)(\gamma : [\![\Gamma]\!](w))(a : [\![A]\!]_{vtm}(w)) = [\![M]\!]_{ctm}(w)(\gamma, a) \\
&[\![M\ N]\!]_{ctm}(w)(\gamma : [\![\Gamma]\!](w), \delta : [\![\Delta]\!](w)) = [\![M]\!]_{ctm}(w)(\gamma)([\![N]\!]_{vtm}(w)(\delta))
\end{aligned}$$

$$\begin{aligned}
[\![\mho]\!]_{ctm} &= \text{add error to alloc monad} \\
[\![\text{force } V]\!]_{ctm} &= \\
[\![\text{ret } V]\!]_{ctm} &= \\
[\![x \leftarrow M;\ N]\!]_{ctm} &= \\
[\![M\ N]\!]_{ctm} &= \\
[\![M@N]\!]_{ctm} &= \\
[\![\lambda x : A.M]\!]_{ctm} &= \\
[\![\alpha x : A.M]\!]_{ctm} &= \\
[\![\text{newcase}_A\ \sigma;\ M]\!]_{ctm} &= \\
[\![\text{match } V_1 \text{ with } V_2\ \{x.M \mid N\}]\!]_{ctm} &= \\
[\![\text{if } V \text{ then } M \text{ else } N]\!]_{ctm} &=
\end{aligned}$$

TODO: notation consistency (ex subscripts on $[\![\,]\!]$)

[18] The computation judgments in Levy's model are not required to satisfy naturality. The problem is the store type, which we don't have (section 6.7.2 paragraph 2 [**?**]), that is not covariant or contravariant in world morphisms.