1 Untyped Syntax

value types	A	$::=$	$Unit$	
		$\mid$	$Bool$	
		$\mid$	$CaseA$	
		$\mid$	$OSum$	
		$\mid$	$A \times A$	
		$\mid$	$A * A$	
		$\mid$	$U\underline{B}$	
computation types	B	$::=$	$A \to \underline{B}$	
		$\mid$	$A \mathrel{-\!*} \underline{B}$	
		$\mid$	$F\ A$	
values	V	$::=$	$\texttt{tt}$	
		$\mid$	true	
		$\mid$	false	
		$\mid$	x	*variable*
		$\mid$	σ	*type tag*
		$\mid$	$\texttt{inj}_V\, V$	*injection into the open sum*
		$\mid$	(V, V)	
		$\mid$	$(V * V)$	
		$\mid$	$\texttt{thunk}\ M$	
computations	M	$::=$	$\mho$	
		$\mid$	$\texttt{print}\ V$	
		$\mid$	$\texttt{force}\ V$	
		$\mid$	$\texttt{ret}\ V$	
		$\mid$	$x \leftarrow M;\ N$	
		$\mid$	$M\ N$	
		$\mid$	$M@N$	
		$\mid$	$\lambda x\colon A.M$	
		$\mid$	$\alpha x\colon A.M$	
		$\mid$	$\texttt{newcase}_A\ x;\ M$	*"allocate" a new case in the open sum*
		$\mid$	$\texttt{match}\ V\ \texttt{with}\ V\{\texttt{inj}\,x.M \mid N\}$	*match on open sum*
		$\mid$	$\texttt{let}\ (x, y) = V;\ M$	
		$\mid$	$\texttt{let}\ (x * y) = V;\ M$	
		$\mid$	$\texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ N$	
value typing context	Γ	$::=$	$x : A$	
		$\mid$	ϕ	*cartesian unit*
		$\mid$	$\Gamma; \Gamma$	*cartesian product*
		$\mid$	φ	*monoidal unit*
		$\mid$	$\Gamma \fatsemi \Gamma$	*monoidal product*
stoup	Δ	$::=$	$\cdot$	
		$\mid$	$\bullet$	

2 Semantics

We will give a semantics to this CBPV calculus using a simple monad algebra model with printing and error effects.

2.1 Value Semantics

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 a world is a map from case symbols to syntactic types. Let Sym denote a set of case symbols $Sym = \{\sigma_{Bool}.\sigma_{\times}, ..\}$. Let $Type^{Syn}$ be a set of syntactic types that can be used in the OSum Case store.

$$\begin{array}{llll} SynTy & Ty & ::= & Unit \\ & & \mid & Bool \\ & & \mid & Ty \times Ty \end{array}$$

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 case symbols 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, [4].

$$World \cong Sym \rightharpoonup_{fin} SemTy \quad SemTy \cong [\textbf{Worlds}, Set]$$

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

Here we define the category $\widehat{\textbf{Worlds}}$. A $world : Sym \rightharpoonup_{fin} SynTy$ is a finite map from case symbols to syntactic types. We can define an order relation on worlds $w \leq w'$[1]:

$$w \leq w' \Leftrightarrow dom(w') \subseteq dom(w) \wedge \forall (x \in dom(w')), w(x) = w'(x)$$

This order is reflexive and transitive, thus we have a preorder on $worlds$ which can be viewed as a thin category which we will denote **Worlds**. We take the category of presheaves over **Worlds**, denoted $\widehat{\textbf{Worlds}}$, to be our "value category". As a presheaf category, $\widehat{\textbf{Worlds}}$ has products, exponents, and coproducts.

[1]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).

2.1.1 Monoidal Structure

We need to introduce additional structure on $\widehat{\textbf{Worlds}}$ to interpret the separating conjunctives $A * B$ and $A \mathrel{-\!\!*} B$. For this, we will pull from existing literature on categorical semantics of bunched typing, specifically section 3.1 of [3]. In essence, we use Day convolution [cite] to get a monoidal structure on $\widehat{\textbf{Worlds}}$ from a promonoidal structure on **Worlds**. First, we define an operation $x *_w y'$ on $worlds$.

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

where

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

The unit for this partial monoid structure is the empty map $\lambda()$[2]. Using this operation[3], we define the monoidal product for $\widehat{\textbf{Worlds}}$:

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

The concrete interpretation for this operation is

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

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

$$(Y, Z, a \in A(Y), b \in B(Z)) \leq (Y', Z', a' \in A(Y'), b' \in B(Z'))$$
if
$$f : Y \leq Y', g : Z \leq Z', a' = A(f)(a), b' = B(g)(b)$$

The monoidal unit is given by:

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

And the separating implication is defined by:

$$(A \mathrel{-\!\!*} B)X = \int_Z \textbf{Set}[A(Z), B(X *_w Z)] \cong \widehat{\textbf{Worlds}}[A, B(X *_w \_)]$$

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 stucture on $\widehat{\textbf{Worlds}}$ along with the monoidal structure and its right adjoint bundled together has the structure of a bicartesian doubly closed category or BiDCC[4].

[2]The 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

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

2.1.2 Affine BiDCC

The concrete model that we are working with has the property that $\mathbf{1} \cong I$[5][6]. That is, the cartesian unit is isomorphic to the monoidal unit. This property validates weakening for $⨟$. Note that

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

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

2.2 Effect Monad

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

$$T_0(X) = Const_E + (X \times Const_{FooBar})$$

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:

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

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 $\otimes$

2.3 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 **TAlg**. **TAlg** has all limits which exist in $\widehat{\mathbf{Worlds}}$[7]. Concrete constructions can be found here [1]. The singleton, exponential, product, and free algebra are ”given”, what remains is to construct the separating implication algebra. (TODO: define)

[5] TODO, work this out in full detail
[6] See section 3.3 of [3]
[7] https://ncatlab.org/nlab/show/Eilenberg-Moore+category

2.4 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 $[\![\_]\!]_{el} : SynTy \to \widehat{\mathbf{Worlds}}$. 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 $\widehat{\mathbf{Worlds}}$.

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

Next, we interpret the remaining value types in $\widehat{\mathbf{Worlds}}$. The separating product is interpreted as the monoidal product from the Day convolution and $\mid [\![B]\!]_c \mid$ 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 $\{\sigma \mid \sigma \in dom(\rho) \land [\![\rho(\sigma)]\!]_{el} \cong [\![A]\!]_{el}\}$. 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 $[\![A]\!] \cong [\![Unit \times A]\!]$?

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

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

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

2.5 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 "$\fatsemi$") and the structure of contexts are "tree-like".

TODO: environment interpretation Use with one or two monoidal units? The interpretation of the separating connectives used here is affine [2].

2.6 Interpreting Terms

$$
\begin{aligned}
[\![-]\!]_{vtm} &: Type^{Syn} \rightarrow \\
[\![\text{foo}]\!]_{vtm}(\gamma) &= foo \\
[\![\text{bar}]\!]_{vtm}(\gamma) &= bar \\
[\![\text{tt}]\!]_{vtm}(\gamma) &= * \\
[\![\text{true}]\!]_{vtm}(\gamma) &= inl\ * \\
[\![\text{false}]\!]_{vtm}(\gamma) &= inr\ * \\
[\![x]\!]_{vtm}(\gamma) &= lookup(\gamma, x)\text{TODO: define for bunches} \\
[\![\sigma]\!]_{vtm}(\gamma) &= \text{stored in gamma, monoidal vs cartesian} \\
[\![\text{inj}_\sigma\ V]\!]_{vtm} &= \text{injection based on match with } \rho(\sigma)\text{?} \\
[\![(V_1, V_2)]\!]_{vtm} &= \\
[\![(V_1 * V_2)]\!]_{vtm} &= \\
[\![\text{thunk } M]\!]_{vtm} &=
\end{aligned}
$$

$$
\begin{aligned}
[\![\mho]\!]_{ctm} &= \\
[\![\text{print } V]\!]_{ctm} &= \\
[\![\text{force } V]\!]_{ctm} &= \\
[\![\text{ret } V]\!]_{ctm} &= \\
[\![x \leftarrow M;\ N]\!]_{ctm} &= \\
[\![M\ N]\!]_{ctm} &= \\
[\![M@N]\!]_{ctm} &= \\
[\![\lambda x : A.M]\!]_{ctm} &= \\
[\![\alpha x : A.M]\!]_{ctm} &= \\
[\![\text{newcase}_A\ \sigma;\ M]\!]_{ctm} &= \text{updated world and value environment?} \\
[\![\text{match } V_1 \text{ with } V_2\{x.M \mid N\}]\!]_{ctm} &= \\
[\![\text{let } (x, y) = V;\ M]\!]_{ctm} &= \\
[\![\text{let } (x * y) = V;\ M]\!]_{ctm} &= \\
[\![\text{if } V \text{ then } M \text{ else } N]\!]_{ctm} &=
\end{aligned}
$$

References

[1] 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.

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

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

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