Adjunction Semantics For Call-By-Push-Value

Paul Blain Levy

School of Computer Science, University of Birmingham, UK

Theory Seminar, 12 March 2004

Overview

Some category theory: modules, products, coproducts and adjunctions

Call-by-push-value and its CK-machine

Example models: domains, store, continuations, . . .

Adjunction semantics

Left Modules

If $\mathcal{C}$ is a category, a left $\mathcal{C}$-module is . . . ?

Pictorially, it's a collection of out-morphisms $B \xrightarrow{\quad g \quad}$

equipped with composition $A \xrightarrow{\quad f \quad} B \xrightarrow{\quad g \quad}$

satisfying associativity and left-identity laws:

$$\begin{aligned} (f'; f); g &= f'; (f; g) \\ \mathsf{id}; g &= g \end{aligned}$$

Abstractly, it's a functor $\mathcal{N} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$.

Right Modules

If $\mathcal{D}$ is a category, a right $\mathcal{D}$-module is . . . ?

Pictorially, it's a collection of in-morphisms $\xrightarrow{\quad g \quad} B$

equipped with composition $\xrightarrow{\quad g \quad} B \xrightarrow{\quad h \quad} C$

satisfying associativity and left-identity laws:

$$\begin{aligned} g;(h;h') &= (g;h);h' \\ \mathrm{id};g &= g \end{aligned}$$

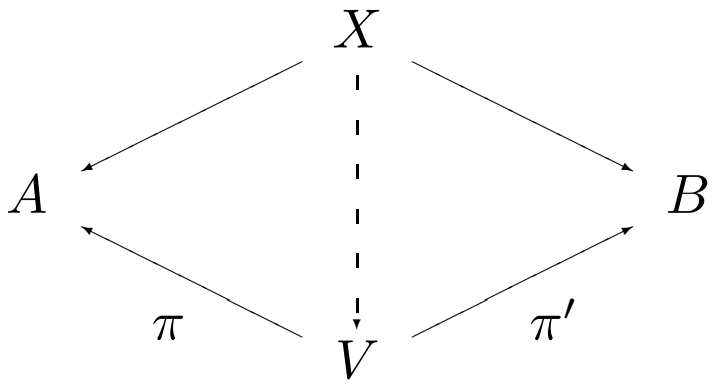
Abstractly, it's a functor $\mathcal{O} : \mathcal{D} \longrightarrow \mathbf{Set}$.

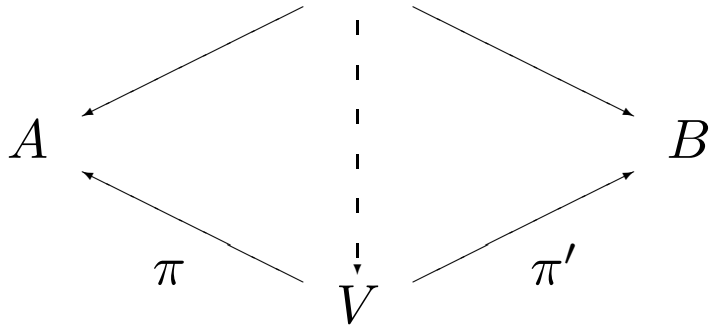
Products In The Presence Of Right Modules

Let $\mathcal{D}$ be a category (underline objects), accompanied by a right module $\mathcal{O}$, or even a family of right modules $\{\mathcal{O}_p\}_{p\in P}$.

For objects A and B, a product in this setting

is an object V and morphisms $A \xleftarrow{\pi} V \xrightarrow{\pi'} B$ such that



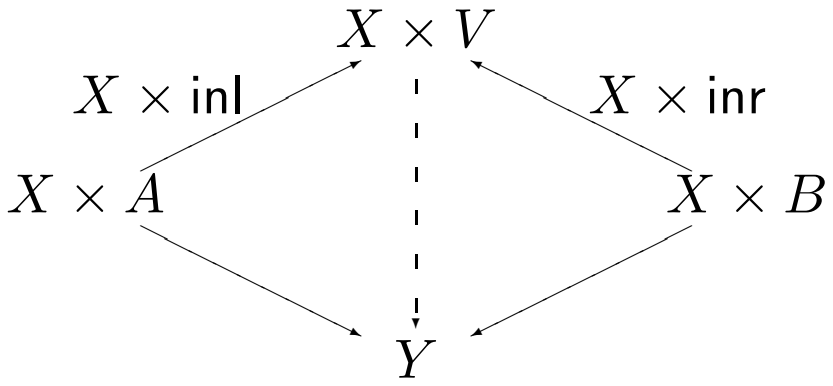


Distributive Coproducts

Let $\mathcal{C}$ be a cartesian category.

For objects A and B, a distributive coproduct

is an object V and morphisms $A \xrightarrow{\mathsf{inl}} V \xleftarrow{\mathsf{inr}} B$ such that



Distributive Coproducts (continued)

Any distributive coproduct in cartesian $\mathcal{C}$ is a coproduct.

Conversely if $\mathcal{C}$ is cartesian closed.

A distributive category is a cartesian category with finite distributive coproducts.

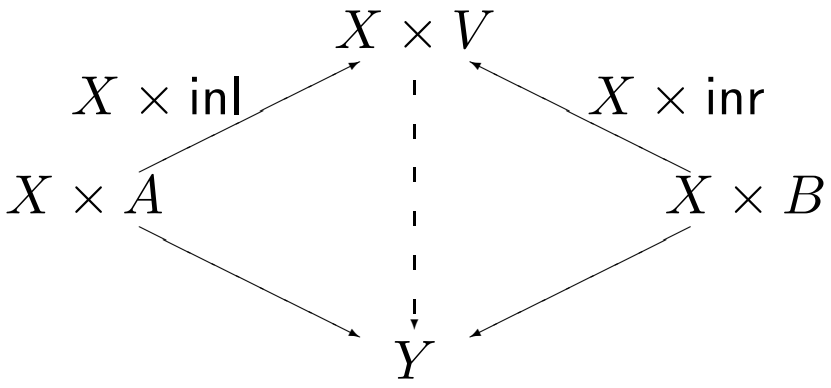
There are several equivalent definitions.

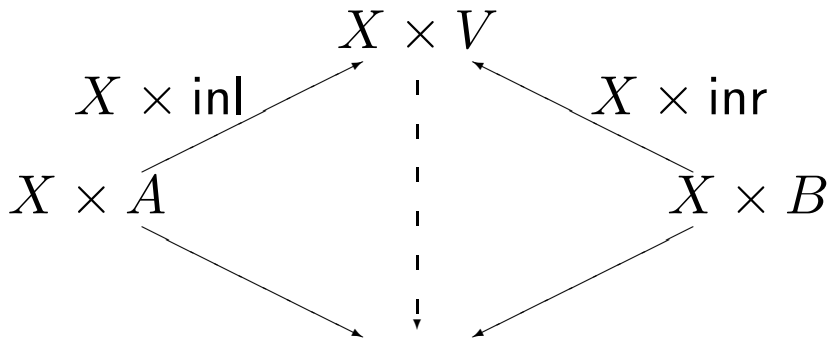
Distributive Coproducts In The Presence Of Left Modules

Let $\mathcal{C}$ be a cartesian category, accompanied by a left module $\mathcal{N}$, or even a family of left modules $\{\mathcal{N}_p\}_{p \in P}$.

For objects A and B, a distributive coproduct in this setting

is an object V and morphisms $A \xrightarrow{\text{inl}} V \xleftarrow{\text{inr}} B$ such that





Bimodules

If $\mathcal{C}$ and $\mathcal{D}$ be categories (underline $\mathcal{D}$-objects), a $(\mathcal{C}, \mathcal{D})$-bimodule is . . . ?

Pictorially it's a collection of oblique morphisms $A \xrightarrow{\quad g \quad} \underline{B}$

equipped with left composition $A \xrightarrow{\quad f \quad} B \xrightarrow{\quad g \quad} \underline{C}$

and right composition $A \xrightarrow{\quad g \quad} \underline{B} \xrightarrow{\quad h \quad} \underline{C}$

satisfying associativity and identity laws, including

$$(f; g); h \quad = \quad f; (g; h)$$

Abstractly, it's a functor $\mathcal{M} : \mathcal{C}^{\mathsf{op}} \times \mathcal{D} \longrightarrow \mathbf{Set}$.

Adjunctions

An adjunction $\mathcal{C} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathcal{D}$ consists of an isomorphism

$$\mathcal{C}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \qquad \text{natural in } X \text{ and } \underline{Y}$$

Alternatively, it's a $(\mathcal{C}, \mathcal{D})$-bimodule, together with isomorphisms

$$\begin{aligned} \mathcal{C}(X, U\underline{Y}) &\cong \mathcal{M}(X, \underline{Y}) \qquad \text{natural in } X \\ \mathcal{M}(X, \underline{Y}) &\cong \mathcal{D}(FX, \underline{Y}) \qquad \text{natural in } \underline{Y} \end{aligned}$$

where U and F are given on objects only.

The Story of Call-By-Push-Value (1)

Study semantics for call-by-value and call-by-name (with η-law):

- cpos for CBV, pointed cpos for CBN
- Moggi's models of CBV using a strong monad
- models of CBN using algebras for the monad
- O'Hearn's storage semantics for CBN
- Streicher-Reus' continuation semantics for CBN

The Story of Call-By-Push-Value (2)

Compare them...a pattern emerges...an underlying set of primitives.

For example,

$S \times -$ in storage semantics

$- \longrightarrow R$ in continuation semantics

appear in exactly the same places.

The pattern is in semantics of types, of judgements and of terms.

Finally, work out the operational meaning of these primitives.

Values and Computations

A term is either a value or a computation.

Intuition A value is, a computation does.

A value has value type, a computation has computation type.

Types

value types $\quad A ::= \quad UB \mid \sum_{i \in I} A_i \mid 1 \mid A \times A$

computation types $\quad B ::= \quad FA \mid \prod_{i \in I} B_i \mid A \to B$

Strangely

Function types are computation types. $\lambda \mathrm{x}.M$ is a computation.

Scott semantics

Value type denotes cpo; computation type denotes pointed cpo.

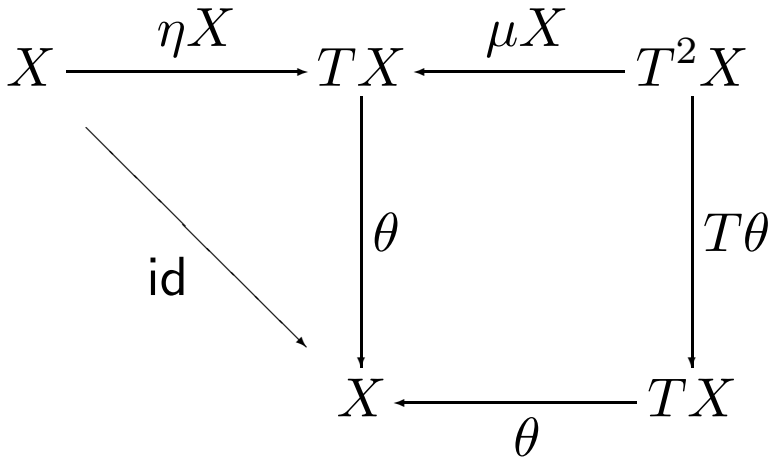
F denotes lift, but U is invisible.

Algebras

An algebra for a monad (T, η, μ) on a category $\mathcal{C}$ is

- an object X (the carrier)
- a morphism $TX \xrightarrow{\theta} X$ (the structure)

such that



commutes.

Algebra Semantics For (Types of) CBPV

Let (T, η, μ, t) be a strong monad on a distributive category $\mathcal{C}$, with sufficient exponentials.

Value type denotes a $\mathcal{C}$-object; computation type denotes a T-algebra.

$U\underline{B}$ denotes carrier of $[\![\underline{B}]\!]$, while FA denotes free algebra on $[\![A]\!]$.

An algebra for the lifting monad on $\mathbf{Cpo}$ is a pointed cpo, so Scott semantics is a special case.

Judgements

An identifier gets bound to a value, so it has value type.

A context Γ is a list of identifiers with value types

$$\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{m-1} : A_{m-1}$$

Judgement for a value: $\Gamma \vdash^{\mathsf{v}} V : A$

Judgement for a computation: $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$

The Returner Type FA

A computation in FA returns a value in A.

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

$$\frac{}{\mathtt{return}\ V \Downarrow \mathtt{return}\ V} \qquad \frac{M \Downarrow \mathtt{return}\ V \quad N[V/\mathtt{x}] \Downarrow T}{M\ \mathtt{to}\ \mathtt{x}.\ N \Downarrow T}$$

This follows Moggi and Filinski.

The Thunk Type $U\underline{B}$

A value in $U\underline{B}$ is a thunk of a computation in $\underline{B}$.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{M \Downarrow T}{\mathtt{force}\ \mathtt{thunk}\ M \Downarrow T}$$

The constructs `thunk` and `force` are inverse. They are invisible in Scott semantics and all monad semantics.

The Function Type $A \to \underline{B}$

A function is a computation. But its argument is a value.

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V \text{`} M : \underline{B}}$$

$$\frac{}{\lambda \mathtt{x}.M \Downarrow \lambda \mathtt{x}.M} \qquad \frac{M \Downarrow \lambda \mathtt{x}.N \quad N[V/\mathtt{x}] \Downarrow T}{V \text{`} M \Downarrow T}$$

We write application the wrong way round, using the ‘ symbol.

Identifiers

An identifier is a value.

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{M[V/\mathtt{x}] \Downarrow T}{\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M \Downarrow T}$$

We write `let` to bind an identifier.

Models of CBPV

Here is the interpretation of CBPV connectives in a number of models:

model	U	F	$\prod_{i\in I}$	$\to$
algebras	U^T	F^T	$\prod_{i\in I}$	$\to$
store	$S \to -$	$S \times -$	$\prod_{i\in I}$	$\to$
continuations	$- \to R$	$- \to R$	$\sum_{i\in I}$	$\times$
erratic choice	$\mathcal{P}$	$-$	$\sum_{i\in I}$	$\times$

Monads Into Adjunctions

These models resolve Moggi's monad into an adjunction.

- monad T into $\mathcal{C}$ $\underset{U^T}{\overset{F^T}{\rightleftarrows}}$ $\mathcal{C}^T$ (category of T-algebras)
- the storage monad into $\mathbf{Set}$ $\underset{S \to -}{\overset{S \times -}{\rightleftarrows}}$ $\mathbf{Set}$
- the continuation monad into $\mathbf{Set}$ $\underset{- \to R}{\overset{- \to R}{\rightleftarrows}}$ $\mathbf{Set}^{\mathrm{op}}$

Where's The Adjunction In CBPV?

An adjunction requires categories $\mathcal{C}$ and $\mathcal{D}$, and a $(\mathcal{C},\mathcal{D})$-bimodule.

$\mathcal{C}$-objects are value types, $\mathcal{D}$-objects are computation types.

$\mathcal{C}$-morphisms are values, oblique morphisms are computations.

But where are the $\mathcal{D}$-morphisms?

The CK-Machine

An operational semantics due to Felleisen-Friedman (1986).

It can be used for CBV, CBN and CBPV.

At any time, there's a computation (C) and a stack of contexts (K).

Initially and finally, K is the empty stack `nil`.

Some authors make K into a single context, called an evaluation context.

Sequencing

The big-step rule $\dfrac{M \Downarrow \texttt{return}\ V \quad N[V/\texttt{x}] \Downarrow T}{M\ \texttt{to x.}\ N \Downarrow T}$ becomes

$$\begin{array}{lll} M\ \texttt{to x.}\ N & K & \rightsquigarrow \\ M & [\cdot]\ \texttt{to x.}\ N :: K & \end{array}$$

$$\begin{array}{lll} \texttt{return}\ V & [\cdot]\ \texttt{to x.}\ N :: K & \rightsquigarrow \\ N[V/\texttt{x}] & K & \end{array}$$

Application

The big-step rule $\dfrac{M \Downarrow \lambda \mathtt{x}.N \quad N[V/\mathtt{x}] \Downarrow T}{V\text{`}M \Downarrow T}$ becomes

$$\begin{array}{lrl} V\text{`}M & K & \rightsquigarrow \\ M & V :: K & \end{array}$$

$$\begin{array}{lrl} \lambda \mathtt{x}.N & V :: K & \rightsquigarrow \\ N[V/\mathtt{x}] & K & \end{array}$$

Functions Pop

$$
\begin{array}{lrcl}
V`M & K & \rightsquigarrow \\
M & V :: K &
\end{array}
$$

$$
\begin{array}{lrcl}
\lambda \mathtt{x}.N & V :: K & \rightsquigarrow \\
N[V/\mathtt{x}] & K &
\end{array}
$$

So $V`$ means push V and $\lambda \mathtt{x}$ means pop $\mathtt{x}$.

A computation in $A \to \underline{B}$ pops a value in A, then behaves in $\underline{B}$.

That's why it's a computation.

Example Program In F nat

```
print "hello0".
let 3 be x.
let  thunk (
        print "hello1".
        λz.
        print "we just popped " z.
        return x + z
     ) be y.
( print "hello2".
  7 ‘
  print "we just pushed 7".
  force y
) to w.
print "w is bound to " w.
return w + 5
```

Subject Reduction: The Typed CK-Machine

Initial Configuration

$$M \quad \underline{C} \qquad \texttt{nil} \quad \underline{C}$$

Transitions

$$\begin{array}{lllll} M \texttt{ to x. } N & \underline{B} & K & \underline{C} & \rightsquigarrow \\ M & FA & [\cdot] \texttt{ to x. } N :: K & \underline{C} & \end{array}$$

$$\begin{array}{lllll} \texttt{return } V & FA & [\cdot] \texttt{ to x. } N :: K & \underline{C} & \rightsquigarrow \\ N[V/\texttt{x}] & \underline{B} & K & \underline{C} & \end{array}$$

Non-Closed Terms

Initial Configuration

$$\Gamma \quad M \quad \underline{C} \qquad\qquad \texttt{nil} \quad \underline{C}$$

Transitions

$$\begin{array}{llll} \Gamma & M \texttt{ to x. } N & \underline{B} & K \quad \underline{C} \quad \rightsquigarrow \\ \Gamma & M & FA & [\cdot] \texttt{ to x. } N :: K \quad \underline{C} \end{array}$$

$$\begin{array}{llll} \Gamma & \texttt{return } V & FA & [\cdot] \texttt{ to x. } N :: K \quad \underline{C} \quad \rightsquigarrow \\ \Gamma & N[V/\texttt{x}] & \underline{B} & K \quad \underline{C} \end{array}$$

Typing Stacks

A configuration takes the form

$$\Gamma \qquad M \qquad \underline{B} \qquad K \qquad \underline{C}$$

We know how to type the computation: $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$

We need a judgement for the stack: $\Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$

Typing Rules

We can read off typing rules from the CK-machine, e.g.

$$\frac{}{\Gamma \mid \underline{C} \vdash^{\mathsf{k}} \mathtt{nil} : \underline{C}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \mid FA \vdash^{\mathsf{k}} [\cdot] \mathtt{\ to\ x.\ } M :: K : \underline{C}}$$

Operations on Stacks (1): Substitution

Following Lawvere, we define a category of contexts.

A morphism from Γ to $\Delta = B_0, \ldots, B_{n-1}$ is a sequence of values $V_0, \ldots, V_{n-1}$ such that $\Gamma \vdash^{\mathsf{v}} V_i : B_i$.

We can substitute such a morphism $\overrightarrow{V_i}$ into a term $\Delta \vdash P$

to obtain a term $\Gamma \vdash \overrightarrow{V_i}^* P$.

$$\begin{aligned} (\overrightarrow{V_i}; \overrightarrow{W_j})^* P &= \overrightarrow{V_i}^* (\overrightarrow{W_j}^* P) \\ \overrightarrow{\mathtt{x}_i}^* P &= P \end{aligned}$$

Operations on Stacks (2): Concatenation

Given two stacks $\Gamma \mid \underline{A} \vdash^{\mathsf{k}} K : \underline{B}$ and $\Gamma \mid \underline{B} \vdash^{\mathsf{k}} L : \underline{C}$,

we can concatenate them to get $\Gamma \mid \underline{A} \vdash^{\mathsf{k}} K +\!\!+ L : \underline{C}$.

$$
\begin{array}{rcl}
(J +\!\!+ K) +\!\!+ L & = & J +\!\!+ (K +\!\!+ L) \\
\texttt{nil} +\!\!+ K & = & K \\
K +\!\!+ \texttt{nil} & = & K
\end{array}
$$

$$
\overrightarrow{V_i}^*(K +\!\!+ L) \quad = \quad (\overrightarrow{V_i}^* K) +\!\!+ (\overrightarrow{V_i}^* L)
$$

Locally $\mathcal{C}$-Indexed Categories

If $\mathcal{C}$ is a category, a locally $\mathcal{C}$-indexed category is . . . ?

Pictorially, it's a collection of objects and morphisms $\underline{B} \xrightarrow[A]{h} \underline{C}$

equipped with composition, identities,

and reindexing along $A' \xrightarrow{f} A$ to give $\underline{B} \xrightarrow[A']{f^*h} \underline{C}$

satisfying associativity, identity and reindexing laws.

Abstractly, it's a strictly $\mathcal{C}$-indexed category $\mathcal{D}$ where all the fibres have the same objects ob $\mathcal{D}$, and all the reindexing functors are identity-on-objects.

Very abstractly, it's a $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$-enriched category.

Examples Of Locally $\mathcal{C}$-Indexed Categories

self $\mathcal{C}$ for cartesian category $\mathcal{C}$. It has the same objects as $\mathcal{C}$, and a morphism $B \xrightarrow[A]{} C$ is a $\mathcal{C}$-morphism $A \times B \longrightarrow C$.

$\mathbf{Cpo}^{\perp}$ is locally $\mathbf{Cpo}$-indexed. An object is a pointed cpo, and a morphism $B \xrightarrow[A]{} C$ is a continuous function $A \times B \longrightarrow C$ strict in its second argument. More generally:

$\mathcal{C}^T$ for a strong monad T. An object is a T-algebra, and a morphism is homomorphic in its second argument.

Operations on stacks (3): Dismantling

A computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and a stack $\Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ give a configuration

$$\Gamma \qquad M \qquad \underline{B} \qquad K \qquad \underline{C}$$

We can dismantle K onto M ("running the CK-machine in reverse") to get $\Gamma \vdash^{\mathsf{c}} M \bullet K : \underline{C}$.

$$\begin{aligned} M \bullet (K +\!\!+ L) &= (M \bullet K) \bullet L \\ M \bullet \mathtt{nil} &= M \end{aligned}$$

$$\overrightarrow{V_i}^*(M \bullet K) = (\overrightarrow{V_i}^* M) \bullet (\overrightarrow{V_i}^* K)$$

Locally Indexed Right $\mathcal{D}$-Modules

If $\mathcal{D}$ is a locally indexed $\mathcal{C}$-category, then a right $\mathcal{D}$-module is a. . . ?

Pictorially it's a collection of in-morphisms $\xrightarrow[A]{g} \underline{B}$

equipped with composition $\xrightarrow[A]{g} \underline{B} \xrightarrow[A]{h} \underline{C}$

and reindexing by $A' \xrightarrow{f} A$ to give $\xrightarrow[A']{f^*g} \underline{B}$

satisfying associativity, right-identity and reindexing laws

Very abstractly, it's a functor $\mathcal{O} : \mathrm{opGr}\mathcal{D} \longrightarrow \mathbf{Set}$.

Equally abstractly, it's a $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$-enriched right $\mathcal{D}$-module.

Semantics of Judgements: A Summary

To interpret judgements of CBPV, we require a

- cartesian category $\mathcal{C}$, for values
- locally $\mathcal{C}$-indexed category $\mathcal{D}$, for stacks
- right $\mathcal{D}$-module $\mathcal{O}$, for computations

We call objects of $\mathcal{C}$ and $\mathcal{D}$ the val-objects and comp-objects respectively.

But how can we model the connectives (other than $1, \times$)?

Right Adjunctives: The U Types

The reversible rule for $U\underline{B}$ is

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{v}} U\underline{B}}$$

So for each comp-object $\underline{B}$ we need a val-object $U\underline{B}$ and an isomorphism

$$\mathcal{O}_X\underline{B} \quad \cong \quad \mathcal{C}(X, U\underline{B}) \qquad\qquad \text{natural in } X$$

This is called a right adjunctive for $\underline{B}$.

Left Adjunctives: The F Types

The reversible rule for FA is

$$\frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma \mid FA \vdash^{\mathsf{k}} \underline{B}}$$

So for each val-object A we need a comp-object FA and an isomorphism

$$\mathcal{O}_{X \times A} \underline{Y} \quad \cong \quad \mathcal{D}_X(FA, \underline{Y}) \qquad \text{natural in } X, \underline{Y}$$

This is called a left adjunctive for A.

Strong Adjunctions

Let $\mathcal{C}$ be a cartesian category, and let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category.

A strong adjunction from $\mathcal{C}$ to $\mathcal{D}$ consists of a right $\mathcal{D}$-module $\mathcal{O}$, with all right adjunctives and all left adjunctives.

A strong adjunction from $\mathcal{C}$ gives rise to a strong monad on $\mathcal{C}$.

2-Categorical Perspective

Locally $\mathcal{C}$-indexed categories form a 2-category.

monad on self $\mathcal{C}$ = strong monad on $\mathcal{C}$.

adjunction from self $\mathcal{C}$ to $\mathcal{D}$ = strong adjunction from $\mathcal{C}$ to $\mathcal{D}$.

And an adjunction from self $\mathcal{C}$ gives a monad on self $\mathcal{C}$.

Exponentials and Products

For $\prod_{i\in I}\underline{B}_i$ and $A \to \underline{B}$, there are 2 reversible rules:

computations

$$\frac{\cdots\ \Gamma \vdash^{\mathsf{c}} \underline{B}_i\ \cdots_{\,i\in I}}{\Gamma \vdash^{\mathsf{c}} \prod_{i\in I}\underline{B}_i} \qquad \frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{c}} A \to \underline{B}}$$

stacks

$$\frac{\cdots\ \Gamma|\underline{C} \vdash^{\mathsf{k}} \underline{B}_i\ \cdots_{\,i\in I}}{\Gamma|\underline{C} \vdash^{\mathsf{k}} \prod_{i\in I}\underline{B}_i} \qquad \frac{\Gamma, A \mid \underline{C} \vdash^{\mathsf{k}} \underline{B}}{\Gamma \mid \underline{C} \vdash^{\mathsf{k}} A \to \underline{B}}$$

Similarly for $\prod_{i\in I}\underline{B}_i$.

Required Isomorphisms

We need isomorphisms

$$\prod_{i \in I} \mathcal{O}_X \underline{B}_i \quad \cong \quad \mathcal{O}_X \prod_{i \in I} \underline{B}_i$$

$$\prod_{i \in I} \mathcal{D}_X(\underline{Y}, \underline{B}_i) \quad \cong \quad \mathcal{D}_X(\underline{Y}, \prod_{i \in I} \underline{B}_i)$$

Adapt the notion of product in the presence of a right module to the locally indexed setting.

The right module is $\mathcal{O}$.

Exponential is similar.

Sum Type

For $\sum_{i \in I} A_i$, there are 3 reversible rules:

values
$$\frac{\cdots \quad \Gamma, A_i \vdash^{\mathsf{v}} B \quad \cdots_{\, i \in I}}{\Gamma, \sum_{i \in I} A_i \vdash^{\mathsf{v}} B}$$

computations
$$\frac{\cdots \quad \Gamma, A_i \vdash^{\mathsf{c}} \underline{B} \quad \cdots_{\, i \in I}}{\Gamma, \sum_{i \in I} A_i \vdash^{\mathsf{c}} \underline{B}}$$

stacks
$$\frac{\cdots \quad \Gamma, A_i \mid \underline{B} \vdash^{\mathsf{k}} \underline{C} \quad \cdots_{\, i \in I}}{\Gamma, \sum_{i \in I} A_i \mid \underline{B} \vdash^{\mathsf{k}} \underline{C}}$$

Distributive Coproduct: Required Isomorphisms

$$\prod_{i\in I} \mathcal{C}(X \times A_i, Y) \quad \cong \quad \mathcal{C}(X \times \textstyle\sum_{i\in I} A_i, Y)$$

$$\prod_{i\in I} \mathcal{O}_{X\times A_i}\underline{Y} \quad \cong \quad \mathcal{O}_{X\times \sum_{i\in I} A_i}\underline{Y}$$

$$\prod_{i\in I} \mathcal{D}_{X\times A_i}(\underline{Y},\underline{Z}) \quad \cong \quad \mathcal{D}_{X\times \sum_{i\in I} A_i}(\underline{Y},\underline{Z})$$

- Use the left module $\mathcal{O}\_\underline{Y}$ for each comp-object $\underline{Y}$
- Use the left module $\mathcal{D}\_(\underline{Y},\underline{Z})$ for each comp-object $\underline{Y}$ and $\underline{Z}$.

Summary

A CBPV adjunction model is a strong adjunction equipped with all exponentials, finite (countable) products, and finite (countable) distributive coproducts—defined wrt the modules.

All of our examples (algebras, storage, continuations etc.) are instances of this.

Further Development

Each connective is modelled as a representing object for a functor. That's unique up to unique isomorphism, so no coherence conditions are required.

2 full sub-adjunctions are given by the Kleisli and co-Kleisli constructions. These model call-by-value and call-by-name respectively.