A tutorial on call-by-push-value

Paul Blain Levy

University of Birmingham

February 20, 2012

Outline

Typed λ-calculus

We consider typed λ-calculus with boolean, function and sum types.

Types

$$A \quad ::= \quad \texttt{bool} \mid A + A \mid A \to A$$

Typing judgement $\Gamma \vdash M : A$

Terms

$$\begin{array}{rcl} M & ::= & \mathtt{x} \mid \texttt{let } M \texttt{ be x. } M \\ & & \mid \texttt{true} \mid \texttt{false} \mid \texttt{match } M \texttt{ as } \{\texttt{true. } M, \texttt{ false. } M\} \\ & & \mid \texttt{inl } M \mid \texttt{inr } M \mid \texttt{match } M \texttt{ as } \{\texttt{inl x. } M, \texttt{ inr x. } M\} \\ & & \mid \lambda \mathtt{x}.M \mid M M \end{array}$$

Equational Laws

We consider the equational theory generated by the $\beta\eta$-laws.

η-law for $A \to B$

Any term $\Gamma \vdash M : A \to B$ can be expanded as

$$\lambda \mathtt{x}.M\mathtt{x}$$

Anything of function type is a λ-abstraction.

η-law for `bool`

Any term $\Gamma, \mathtt{z} : \mathtt{bool} \vdash M : B$ can be expanded as

$$\mathtt{match}\ \mathtt{z}\ \mathtt{as}\ \{\mathtt{true}.\ M[\mathtt{true}/\mathtt{z}],\ \mathtt{false}.\ M[\mathtt{false}/\mathtt{z}]\}$$

Anything of boolean type is a boolean.

The η-law for sum types is similar.

Denotational semantics in Set

A type denotes a set.

$$
\begin{array}{rcl}
[\![\texttt{bool}]\!] & = & \mathbb{B} \stackrel{\text{def}}{=} \{\mathsf{true}, \mathsf{false}\} \\
[\![A + B]\!] & = & [\![A]\!] + [\![B]\!] \\
[\![A \to B]\!] & = & [\![A]\!] \to [\![B]\!]
\end{array}
$$

A term $\Gamma \vdash M : B$ denotes a function $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} [\![B]\!]$.

Substitution Lemma

Given terms $\Gamma, \mathtt{x} : A \vdash M : B$ and $\Gamma \vdash N : A$

we can obtain $[\![M[N/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![N]\!]$. It is

$$\rho \longmapsto [\![M]\!](\rho, \mathtt{x} \mapsto [\![N]\!]\rho)$$

Corollary

The denotational semantics validates the β and η laws.

Call-by-name evaluation of a closed term

In CBN the terminals are $\texttt{true}, \texttt{false}, \texttt{inl}\ M, \texttt{inr}\ M, \lambda\mathtt{x}.M$
To evaluate

- $\texttt{true}$: return $\texttt{true}$.
- $\lambda\mathtt{x}.M$: return $\lambda\mathtt{x}.M$.
- $\texttt{inl}\ M$: return $\texttt{inl}\ M$.
- $\texttt{let}\ M\ \texttt{be}\ \mathtt{x}.\ N$: evaluate $N[M/\mathtt{x}]$.
- $\texttt{match}\ M\ \texttt{as}\ \{\texttt{true}.\ N,\ \texttt{false}.\ N'\}$: evaluate M. If it returns $\texttt{true}$, evaluate N, but if it returns $\texttt{false}$, evaluate N'.
- $\texttt{match}\ M\ \texttt{as}\ \{\texttt{inl}\ \mathtt{x}.\ N,\ \texttt{inr}\ \mathtt{x}.\ N'\}$: evaluate M. If it returns $\texttt{inl}\ P$, evaluate $N[P/\mathtt{x}]$, but if it returns $\texttt{inr}\ P$, evaluate $N'[P/\mathtt{x}]$.
- $M\,N$: evaluate M. If it returns $\lambda\mathtt{x}.P$, evaluate $P[N/\mathtt{x}]$.

Call-by-value evaluation of a closed term

CBV terminals $T ::= \ \texttt{true} \mid \texttt{false} \mid \texttt{inl}\ T \mid \texttt{inr}\ T \mid \lambda\mathtt{x}.M$
To evaluate

- $\texttt{true}$: return $\texttt{true}$.
- $\lambda\mathtt{x}.M$: return $\lambda\mathtt{x}.M$.
- $\texttt{inl}\ M$: evaluate M. If it returns T, return $\texttt{inl}\ T$.
- $\texttt{let}\ M\ \texttt{be}\ \mathtt{x}.\ N$: evaluate M. If it returns T, evaluate $N[T/\mathtt{x}]$.
- $\texttt{match}\ M\ \texttt{as}\ \{\texttt{true}.\ N,\ \texttt{false}.\ N'\}$: evaluate M. If it returns $\texttt{true}$, evaluate N, but if it returns $\texttt{false}$, evaluate N'.
- $\texttt{match}\ M\ \texttt{as}\ \{\texttt{inl}\ \mathtt{x}.\ N,\ \texttt{inr}\ \mathtt{x}.\ N'\}$: evaluate M. If it returns $\texttt{inl}\ T$, evaluate $N[T/\mathtt{x}]$, but if it returns $\texttt{inr}\ T$, evaluate $N'[T/\mathtt{x}]$.
- $M N$: evaluate M. If it returns $\lambda\mathtt{x}.P$, evaluate N. If that returns T, evaluate $P[T/\mathtt{x}]$.

Adding computational effects

Errors

Let $E = \{\mathrm{CRASH}, \mathrm{BANG}, \mathrm{WALLOP}\}$ be a set of "errors". We add

$$\frac{}{\Gamma \vdash \texttt{error}\ e : B}\ e \in E$$

To evaluate $\texttt{error}\ e$: halt with error message e.

Printing

Let $\mathcal{A} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$ be a set of "characters". We add

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \texttt{print}\ c.\ M : B}\ c \in \mathcal{A}$$

To evaluate $\texttt{print}\ c.\ M$: print c and then evaluate M.

Exercises

1. Evaluate
$$\texttt{let}\ (\texttt{error}\ \mathrm{CRASH})\ \texttt{be}\ \texttt{x}.\ 5$$
in CBV and CBN
2. Evaluate
$$(\lambda \mathrm{x}.(\mathrm{x}+\mathrm{x}))(\texttt{print}\ \texttt{"hello"}.\ 4)$$
in CBV and CBN.
3. Evaluate
$$\begin{array}{l}\texttt{match}\ (\texttt{print}\ \texttt{"hello"}.\ \texttt{inr}\ \texttt{error}\ \mathrm{CRASH})\ \texttt{as} \\ \quad \{\texttt{inl}\ \mathrm{x}.\ \mathrm{x}+1,\ \texttt{inr}\ \mathrm{y}.\ 5\}\end{array}$$
in CBV and CBN.

Big-Step Operational Semantics

We convert our CBV and CBN interpreters into big-step semantics, defined inductively.

- **no effects** We define a relation $M \Downarrow T$ meaning M evaluates to T.
- **errors** We define a relation $M \Downarrow T$ meaning M evaluates to T, and a relation $M \lightning e$ meaning M raises error e.
- **printing** We define a relation $M \Downarrow m, T$ meaning M prints $m \in \mathcal{A}^*$ and finally evaluates to T.

For example, in the case of printing we have rules such as

$$\frac{}{\texttt{true} \Downarrow \varepsilon, \texttt{true}} \qquad \frac{M \Downarrow m, \texttt{true} \qquad N \Downarrow m', T}{\texttt{match } M \texttt{ as } \{\texttt{true}.\ N,\ \texttt{false}.\ N'\} \Downarrow mm', T}$$

These are proved deterministic and total using Tait's method.

Observational equivalence

Two terms $\Gamma \vdash M, M' : B$ are observationally equivalent

when $\mathcal{C}[M]$ and $\mathcal{C}[M']$ have the same behaviour

for every ground (i.e. boolean) context $\mathcal{C}[\cdot]$.

Same behaviour means: print the same string, raise the same error, return the same boolean.

We write $M \simeq_{\mathbf{CBV}} M'$ and $M \simeq_{\mathbf{CBN}} M'$.

The η-law for boolean type: has it survived?

η-law for `bool`

Any term $\Gamma, \mathtt{z} : \mathtt{bool} \vdash M : B$ can be expanded as

$$\mathtt{match\ z\ as}\ \{\mathtt{true}.\ M[\mathtt{true}/\mathtt{z}],\ \mathtt{false}.\ M[\mathtt{false}/\mathtt{z}]\}$$

Anything of boolean type is a boolean.

This holds in CBV, because `z` can only be replaced by `true` or `false`.

But it's broken in CBN, because `z` might raise an error. For example,

$$\mathtt{true} \not\simeq_{\mathbf{CBN}} \mathtt{match\ z\ as}\ \{\mathtt{true}.\ \mathtt{true},\ \mathtt{false}.\ \mathtt{true}\}$$

because we can apply the context

$$\mathtt{let\ error}\ \mathrm{CRASH}\ \mathtt{be\ z.}\ [\cdot]$$

Similarly the η-law for sum types is valid in CBV but not in CBN.

The η-law for functions: has it survived?

η-law for $A \to B$

Any term $\Gamma \vdash M : A \to B$ can be expanded as

$$\lambda \mathtt{x}.M\mathtt{x}$$

Anything of function type is a function.

This fails in CBV, but it holds in CBN.
Similarly

$$\begin{array}{rcl} \lambda \mathtt{x}.\ \mathtt{error}\ e & \simeq_{\mathbf{CBN}} & \mathtt{error}\ e \\ \lambda \mathtt{x}.\ \mathtt{print}\ c.\ M & \simeq_{\mathbf{CBN}} & \mathtt{print}\ c.\ \lambda \mathtt{x}.\ M \end{array}$$

Yet the two sides have different operational behaviour! What's going on?

In CBN, a function gets evaluated only by being applied.

Summary

The pure calculus satisfies all the β- and η-laws.

With computational effects,

- CBV satisfies η for boolean and sum types, but not function types
- CBN satisfies η for function types, but not boolean and sum types.

We want denotational semantics that validate the appropriate η-laws.

We'll do CBV first, as it's easier.

Denotational Semantics of CBV (Moggi)

Take a (strong) monad T on $\mathbf{Set}$.

- For errors: $- + E$
- For printing: $\mathcal{A}^* \times -$

Each type denotes a set (think: the set of terminals)

$$\begin{aligned} [\![\texttt{bool}]\!] &= \mathbb{B} \\ [\![A + B]\!] &= [\![A]\!] + [\![B]\!] \\ [\![A \to B]\!] &= [\![A]\!] \to T[\![B]\!] \end{aligned}$$

Each term $\Gamma \vdash M : B$ denotes a Kleisli morphism,

i.e. a function $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} T[\![B]\!]$.

To prove the soundness of the denotational semantics, we need a substitution lemma.

CBV Substitution Lemma: What Doesn't Work

Can we obtain $[\![M[N/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![N]\!]$?

CBV Substitution Lemma: What Doesn't Work

Can we obtain $[\![M[N/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![N]\!]$? Not in CBV.

CBV Substitution Lemma: What Doesn't Work

Can we obtain $[\![M[N/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![N]\!]$? Not in CBV.

Example that rules out a general substitution lemma

Define $\mathtt{x} : \mathtt{bool} \vdash M, M' : \mathtt{bool}$ and $\vdash N : \mathtt{bool}$

$$
\begin{array}{rcl}
M & \stackrel{\text{def}}{=} & \mathtt{true} \\
M' & \stackrel{\text{def}}{=} & \mathtt{match}\ \mathtt{x}\ \mathtt{as}\ \{\mathtt{true}.\ \mathtt{true},\ \mathtt{false}.\ \mathtt{true}\} \\
N & \stackrel{\text{def}}{=} & \mathtt{error}\ \mathrm{CRASH} \\
[\![M]\!] & = & [\![M']\!] \qquad \text{because } M =_{\eta\,\mathtt{bool}} M' \\
[\![M[N/\mathtt{x}]]\!] & \neq & [\![M'[N/\mathtt{x}]]\!]
\end{array}
$$

CBV Substitution Lemma: What Doesn't Work

Can we obtain $[\![M[N/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![N]\!]$? Not in CBV.

Example that rules out a general substitution lemma

Define $\mathtt{x} : \mathtt{bool} \vdash M, M' : \mathtt{bool}$ and $\vdash N : \mathtt{bool}$

$$\begin{array}{rcl} M & \stackrel{\text{def}}{=} & \mathtt{true} \\ M' & \stackrel{\text{def}}{=} & \mathtt{match}\ \mathtt{x}\ \mathtt{as}\ \{\mathtt{true}.\ \mathtt{true},\ \mathtt{false}.\ \mathtt{true}\} \\ N & \stackrel{\text{def}}{=} & \mathtt{error}\ \mathrm{CRASH} \\ [\![M]\!] & = & [\![M']\!] \qquad \text{because } M =_{\eta\,\mathtt{bool}} M' \\ [\![M[N/\mathtt{x}]]\!] & \neq & [\![M'[N/\mathtt{x}]]\!] \end{array}$$

But we can give a lemma for the substitution of values:

$$V ::= \ \mathtt{true} \mid \mathtt{false} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda\mathtt{x}.M \mid \mathtt{x}$$

The terminals are the closed values.

Substitution Lemma For Values

Each value $\Gamma \vdash V : B$ denotes a function $[\![\Gamma]\!] \xrightarrow{[\![V]\!]^{\mathsf{val}}} [\![B]\!]$ such that

$$\begin{tikzcd}
[\![\Gamma]\!] \arrow[r, "{[\![V]\!]^{\mathsf{val}}}"] \arrow[dr, "{[\![V]\!]}"'] & {[\![B]\!]} \arrow[d, "{\eta[\![B]\!]}"] \\
 & T[\![B]\!]
\end{tikzcd}$$

commutes.

Substitution Lemma

Given a term $\Gamma, \mathtt{x} : A \vdash M : B$ and a value $\Gamma \vdash V : A$

we can obtain $[\![M[V/\mathtt{x}]]\!]$ from $[\![M]\!]$ and $[\![V]\!]^{\mathsf{val}}$. It is

$$\rho \longmapsto [\![M]\!](\rho, \mathtt{x} \mapsto [\![V]\!]^{\mathsf{val}} \rho)$$

Soundness of CBV Denotational Semantics

Errors

- If $M \Downarrow V$ then $[\![M]\!]\varepsilon = \mathsf{inl}\ ([\![V]\!]^{\mathsf{val}}\varepsilon)$.
- If $M \lightning e$ then $[\![M]\!]\varepsilon = \mathsf{inr}\ e$.

Printing

- If $M \Downarrow m, V$ then $[\![M]\!]\varepsilon = \langle m, [\![V]\!]^{\mathsf{val}}\varepsilon\rangle$.

These are straightforward inductions, using the substitution lemma.

Naive Attempt At CBN: “Carrier” Semantics

Each type denotes a set (think: the set of closed terms).
For example $\texttt{bool} \to (\texttt{bool} \to \texttt{bool})$ should denote $T\mathbb{B} \to (T\mathbb{B} \to T\mathbb{B})$.
We define

$$\begin{array}{rcl} [\![\texttt{bool}]\!] & = & T\mathbb{B} \\ [\![A + B]\!] & = & T([\![A]\!] + [\![B]\!]) \\ [\![A \to B]\!] & = & [\![A]\!] \to [\![B]\!] \end{array}$$

Each term $\Gamma \vdash M : B$ should denote a function $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} [\![B]\!]$.

Carrier Semantics: What Goes Wrong

$$\frac{}{\Gamma \vdash \texttt{error}\ e : B}$$

denotes $\rho \mapsto$?

Carrier Semantics: What Goes Wrong

$$\frac{}{\Gamma \vdash \texttt{error}\ e : B}$$ denotes $\rho \mapsto$?

Example:

- suppose $B = \texttt{bool} \to (\texttt{bool} \to \texttt{bool})$
- then B denotes $(\mathbb{B} + E) \to ((\mathbb{B} + E) \to (\mathbb{B} + E))$
- and $\texttt{error}\ e \simeq_{\mathbf{CBN}} \lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \texttt{error}\ e$
- so the answer should be $\lambda x.\ \lambda y.\ \mathsf{inr}\ e$.

Intuition: go down through the function types until we hit a boolean or sum type.

Carrier Semantics: What Goes Wrong

$$\frac{}{\Gamma \vdash \texttt{error}\ e : B}$$

denotes $\rho \mapsto$?

Example:

- suppose $B = \texttt{bool} \to (\texttt{bool} \to \texttt{bool})$
- then B denotes $(\mathbb{B} + E) \to ((\mathbb{B} + E) \to (\mathbb{B} + E))$
- and $\texttt{error}\ e \simeq_{\mathbf{CBN}} \lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{error}\ e$
- so the answer should be $\lambda x.\ \lambda y.\ \mathsf{inr}\ e$.

Intuition: go down through the function types until we hit a boolean or sum type.

A similar problem arises with `match`.

E-pointed semantics of CBN types

A CBN type should denote a set X (the carrier) with some designated elements $E \xrightarrow{\text{error}} X$.

This is called an E-pointed set.

Thus `bool` denotes $\mathbb{B} + E$ with $e \mapsto \text{inr } e$.

If $[\![A]\!] = (X, \text{error})$ and $[\![B]\!] = (Y, \text{error}')$,

- then $A + B$ denotes $(X + Y) + E$ with $e \mapsto \text{inr } e$
- and $A \to B$ denotes $X \to Y$ with $e \mapsto \lambda x.\ \text{error}'(e)$.

E-pointed semantics of CBN types

A CBN type should denote a set X (the carrier) with some designated elements $E \xrightarrow{\text{error}} X$.

This is called an E-pointed set.

Thus `bool` denotes $\mathbb{B} + E$ with $e \mapsto \text{inr } e$.

If $[\![A]\!] = (X, \text{error})$ and $[\![B]\!] = (Y, \text{error}')$,

- then $A + B$ denotes $(X + Y) + E$ with $e \mapsto \text{inr } e$
- and $A \to B$ denotes $X \to Y$ with $e \mapsto \lambda x.\ \text{error}'(e)$.

Can we generalize the notion of E-pointed set to other monads on $\mathbf{Set}$?

Algebras for a Monad

An *Eilenberg-Moore algebra* for a monad T on $\mathbf{Set}$ is

- a set X (the carrier)
- a function $TX \xrightarrow{\theta} X$ (the structure)

satisfying

$$\begin{tikzcd} X \arrow[r, "\eta X"] \arrow[dr, "\mathsf{id}"'] & TX \arrow[d, "\theta"] & T^2X \arrow[l, "\mu X"'] \arrow[d, "T\theta"] \\ & X & TX \arrow[l, "\theta"] \end{tikzcd}$$

Examples of Algebras

An algebra for the $- + E$ monad is an E-pointed set.

Examples of Algebras

An algebra for the $- + E$ monad is an E-pointed set.

An algebra for $\mathcal{A}^* \times -$ is an $\mathcal{A}$-set

i.e. a set X together with a function $\mathcal{A} \times X \xrightarrow{*} X$.

This is what we need to interpret

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \texttt{print}\ c.\ M : B}\, c \in \mathcal{A}$$

If B denotes $(X, *)$ then $\texttt{print}\ c.\ M$ denotes $\rho \mapsto c * (\llbracket M \rrbracket \rho)$

3 Ways Of Building Algebras

Free Algebras

Given a set X, the free T-algebra on X has carrier TX and structure μX.

Product Algebras

Given a family of T-algebras (X_i, θ_i), the product algebra $\prod_{i \in I}(X_i, \theta_i)$ has carrier $\prod_{i \in I} X_i$ and structure given pointwise.

Exponential Algebras

Given a set A and a T-algebra (X, θ), the exponential algebra $A \to (X, \theta)$ has carrier $A \to X$ and structure given pointwise.

Algebra Semantics For CBN Types

Let T be a monad on $\mathbf{Set}$.

A type denotes a T-algebra.

- `bool` denotes the free algebra on $\mathbb{B}$
- If $[\![A]\!] = (X, \theta)$ and $[\![B]\!] = (Y, \phi)$
  - then $A + B$ denotes the free algebra on $X + Y$
  - and $A \to B$ denotes the exponential algebra $X \to (Y, \phi)$.

Algebra semantics for CBN terms

Suppose B denotes the algebra (Y, θ).

Then a term $\Gamma \vdash M : B$ denotes a function between the carrier sets $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} Y$.

$$\frac{\Gamma \vdash M : \texttt{bool} \quad \Gamma \vdash N : B \quad \Gamma \vdash N' : B}{\Gamma \vdash \texttt{match}\ M\ \texttt{as}\ \{\texttt{true}.\ N,\ \texttt{false}.\ N'\} : B}$$

This term denotes

$$\begin{tikzcd}
{[\![\Gamma]\!]} \arrow[d, "{\langle \mathsf{id}, [\![M]\!] \rangle}"'] & & Y \\
{[\![\Gamma]\!] \times T\mathbb{B}} \arrow[r, "t_{[\![\Gamma]\!],\mathbb{B}}"'] & T([\![\Gamma]\!] \times \mathbb{B}) \arrow[r, "{T[[\![N]\!],[\![N']\!]]}"'] & TY \arrow[u, "\theta"']
\end{tikzcd}$$

Soundness of algebra semantics for CBN

Errors

- If $M \Downarrow T : B$ then $[\![M]\!]\varepsilon = [\![T]\!]\varepsilon$
- If $M \not\Downarrow e : B$ then $[\![M]\!]\varepsilon = \text{error } e$ where $[\![B]\!] = (X, \text{error})$

Printing

- If $M \Downarrow m, T : B$ then $[\![M]\!]\varepsilon = m * *([\![T]\!]\varepsilon)$ where $[\![B]\!] = (X, *)$

Straightforward inductive proofs using the substitution lemma.

Summary

We have a denotational semantics for errors and printing for CBV and CBN, and shown their correctness.

Summary

We have a denotational semantics for errors and printing for CBV and CBN, and shown their correctness.

These are instances of a general recipe using a monad T on $\mathbf{Set}$ and its algebras.

Summary

We have a denotational semantics for errors and printing for CBV and CBN, and shown their correctness.

These are instances of a general recipe using a monad T on $\mathbf{Set}$ and its algebras.

A CBV type denotes a set; a CBN type denotes a T-algebra.

Summary

We have a denotational semantics for errors and printing for CBV and CBN, and shown their correctness.

These are instances of a general recipe using a monad T on $\mathbf{Set}$ and its algebras.

A CBV type denotes a set; a CBN type denotes a T-algebra.

They are fundamentally different things.

Semantics of Types, Again

We write $F^T X$ for the free T-algebra (TX, μ_X) on X

Semantics of Types, Again

We write $F^T X$ for the free T-algebra (TX, μ_X) on X

and $U^T(X, \theta)$ for the carrier X of a T-algebra (X, θ).

Semantics of Types, Again

We write $F^T X$ for the free T-algebra (TX, μ_X) on X

and $U^T(X, \theta)$ for the carrier X of a T-algebra (X, θ).

Our CBN semantics of types can be written

$$\begin{array}{rcl} [\![\texttt{bool}]\!] & = & F^T(1+1) \\ [\![A+B]\!] & = & F^T(U^T[\![A]\!] + U^T[\![B]\!]) \\ [\![A \to B]\!] & = & U^T[\![A]\!] \to [\![B]\!] \end{array}$$

Semantics of Types, Again

We write $F^T X$ for the free T-algebra (TX, μ_X) on X

and $U^T(X, \theta)$ for the carrier X of a T-algebra (X, θ).

Our CBN semantics of types can be written

$$\begin{array}{rcl} [\![\texttt{bool}]\!] & = & F^T(1+1) \\ [\![A+B]\!] & = & F^T(U^T[\![A]\!] + U^T[\![B]\!]) \\ [\![A \to B]\!] & = & U^T[\![A]\!] \to [\![B]\!] \end{array}$$

And our CBV semantics of types can be written

$$\begin{array}{rcl} [\![\texttt{bool}]\!] & = & 1+1 \\ [\![A+B]\!] & = & [\![A]\!] + [\![B]\!] \\ [\![A \to B]\!] & = & U^T([\![A]\!] \to F^T[\![B]\!]) \end{array}$$

Call-By-Push-Value Types

Call-by-push-value has

- value types which (like CBV types) denote sets
- computation types which (like CBN types) denote T-algebras.

Call-By-Push-Value Types

Call-by-push-value has

- value types which (like CBV types) denote sets
- computation types which (like CBN types) denote T-algebras.

$$\text{value type} \quad A ::= \; U\underline{B} \mid 1 \mid A \times A \mid 0 \mid A + A \mid \sum_{i \in \mathbb{N}} A_i$$

$$\text{computation type} \quad \underline{B} ::= \; FA \mid A \to \underline{B} \mid 1_{\Pi} \mid \underline{B} \,\Pi\, \underline{B} \mid \prod_{i \in \mathbb{N}} \underline{B}_i$$

Call-By-Push-Value Types

Call-by-push-value has

- value types which (like CBV types) denote sets
- computation types which (like CBN types) denote T-algebras.

value type $\quad A ::= \; U\underline{B} \mid 1 \mid A \times A \mid 0 \mid A + A \mid \sum_{i \in \mathbb{N}} A_i$

computation type $\quad \underline{B} ::= \; FA \mid A \rightarrow \underline{B} \mid 1_\Pi \mid \underline{B} \,\Pi\, \underline{B} \mid \prod_{i \in \mathbb{N}} \underline{B}_i$

Strangely function types are computation types, and $\lambda \mathrm{x}.M$ is a computation.

Judgements

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

Judgements

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

A context Γ is a finite set of identifiers with associated value type

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

Judgements

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

A context Γ is a finite set of identifiers with associated value type

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

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

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

Judgements

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

A context Γ is a finite set of identifiers with associated value type

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

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

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

- A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function $[\![\Gamma]\!] \xrightarrow{[\![V]\!]} [\![A]\!]$
- If $\underline{B}$ denotes (X, θ), then a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} X$.

Note From the viewpoint of monad/algebra semantics, there is no difference between a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and a value $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$.

F and U

The type FA

A computation in FA returns a value in A.

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

F and U

The type FA

A computation in FA returns a value in A.

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

The denotation of $M\ \mathtt{to}\ \mathtt{x}.\ N$ uses the structure of $[\![\underline{B}]\!]$.

The type $U\underline{B}$

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

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

The constructs `thunk` and `force` are inverse.
They are invisible in monad/algebra semantics.

Identifiers

An identifier is a value.

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

We write let to bind an identifier.

Tuples

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \langle \hat{\imath}, V \rangle : \sum_{i \in I} A_i}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B}\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \mathtt{match}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i \in I} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V' \rangle : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle . M : \underline{B}}$$

The rules for 1 are similar.

Functions

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

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i \;\; (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \lambda\{i.M_i\}_{i \in I} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} M\hat{\imath} : \underline{B}_{\hat{\imath}}} \; \hat{\imath} \in I$$

Functions

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

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i \ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \lambda\{i.M_i\}_{i \in I} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} M\hat{\imath} : \underline{B}_{\hat{\imath}}}\ \hat{\imath} \in I$$

It is often convenient to write applications operand-first, as $V`M$ and $\hat{\imath}`M$.

Interpreter

The terminals are computations:

$$\texttt{return}\ V \qquad \lambda\mathtt{x}.M \qquad \lambda\{i.M_i\}_{i \in I}$$

Interpreter

The terminals are computations:

$$\texttt{return}\ V \qquad \lambda\mathtt{x}.M \qquad \lambda\{i.M_i\}_{i\in I}$$

To evaluate

- $\texttt{return}\ V$: return $\texttt{return}\ V$.
- $M\ \texttt{to}\ \mathtt{x}.\ N$: evaluate M. If it returns $\texttt{return}\ V$, then evaluate $N[V/\mathtt{x}]$.
- $\lambda\mathtt{x}.N$: return $\lambda\mathtt{x}.N$.
- MV: evaluate M. If it returns $\lambda\mathtt{x}.N$, evaluate $N[V/\mathtt{x}]$.
- $\lambda\{i.N_i\}_{i\in I}$: return $\lambda\{i.N_i\}_{i\in I}$.
- $M\hat{\imath}$: evaluate M. If it returns $\lambda\{i.N_i\}_{i\in I}$, evaluate $N_{\hat{\imath}}$.
- $\texttt{let}\ V\ \texttt{be}\ \mathtt{x}.\ M$: evaluate $M[V/\mathtt{x}]$.
- $\texttt{force}\ \texttt{thunk}\ M$: evaluate M.
- $\texttt{match}\ \langle\hat{\imath}, V\rangle\ \texttt{as}\ \{\langle i,\mathtt{x}\rangle.M_i\}_{i\in I}$: evaluate $M_{\hat{\imath}}[V/\mathtt{x}]$.
- $\texttt{match}\ \langle V, V'\rangle\ \texttt{as}\ \langle\mathtt{x},\mathtt{y}\rangle.M$: evaluate $M[V/\mathtt{x}, V'/\mathtt{y}]$.

Decomposing CBV into CBPV

A CBV type translates into a value type.

$$A \to B \quad \mapsto \quad U(A \to FB)$$

A CBV term $\mathtt{x} : A, \mathtt{y} : B \vdash M : C$ translates as $\mathtt{x} : A, \mathtt{y} : B \vdash^{\mathsf{c}} M : FC$.

$$\begin{array}{rcl}
\mathtt{x} & \mapsto & \mathtt{return}\ \mathtt{x} \\
\lambda\mathtt{x}.\ M & \mapsto & \mathtt{return}\ \mathtt{thunk}\ \lambda\mathtt{x}.\ M \\
M\ \ N & \mapsto & M\ \mathtt{to}\ \mathtt{f}.\ N\ \mathtt{to}\ \mathtt{y}.\ ((\mathtt{force}\ \mathtt{f})\ \mathtt{y}) \\
\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N & \mapsto & M\ \mathtt{to}\ \mathtt{y}.\ \mathtt{let}\ \mathtt{y}\ \mathtt{be}\ \mathtt{x}.\ N
\end{array}$$

Decomposing CBV into CBPV

A CBV type translates into a value type.

$$A \to B \quad \mapsto \quad U(A \to FB)$$

A CBV term $\mathtt{x} : A, \mathtt{y} : B \vdash M : C$ translates as $\mathtt{x} : A, \mathtt{y} : B \vdash^{\mathsf{c}} M : FC$.

$$\begin{array}{rcl}
\mathtt{x} & \mapsto & \mathtt{return\ x} \\
\lambda\mathtt{x}.\ M & \mapsto & \mathtt{return\ thunk}\ \lambda\mathtt{x}.\ M \\
M\ \ N & \mapsto & M\ \mathtt{to\ f.}\ N\ \mathtt{to\ y.}\ ((\mathtt{force\ f})\ \mathtt{y}) \\
\mathtt{let}\ M\ \mathtt{be\ x.}\ N & \mapsto & M\ \mathtt{to\ y.\ let\ y\ be\ x.}\ N \\
\text{or} & \mapsto & M\ \mathtt{to\ x.}\ N
\end{array}$$

Decomposing CBN into CBPV

A CBN type translates into a computation type.

$$\begin{array}{rcl} \texttt{bool} & \mapsto & F(1+1) \\ \underline{A} + \underline{B} & \mapsto & F(U\underline{A} + U\underline{B}) \\ \underline{A} \to \underline{B} & \mapsto & U\underline{A} \to \underline{B} \end{array}$$

A CBN term $\mathtt{x} : \underline{A}, \mathtt{y} : \underline{B} \vdash M : \underline{C}$ translates as $\mathtt{x} : U\underline{A}, \mathtt{y} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{C}$.

$$\begin{array}{rcl} \mathtt{x} & \mapsto & \texttt{force}\,\mathtt{x} \\ \texttt{let}\ M\ \texttt{be x.}\ N & \mapsto & \texttt{let}\ (\texttt{thunk}\ M)\ \texttt{be x.}\ N \\ \lambda\mathtt{x}.\ M & \mapsto & \lambda\mathtt{x}.\ M \\ M\ N & \mapsto & M\ (\texttt{thunk}\ N) \\ \texttt{inl}\ M & \mapsto & \texttt{return inl thunk}\ M \end{array}$$

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

The translations from CBV and CBN into CBPV preserve these semantics.

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

The translations from CBV and CBN into CBPV preserve these semantics.

Moggi's TA is UFA.

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

The translations from CBV and CBN into CBPV preserve these semantics.

Moggi's TA is UFA.

But

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

The translations from CBV and CBN into CBPV preserve these semantics.

Moggi's TA is UFA.

But

- the monad/algebra semantics makes `thunk` and `force` invisible

Summary

We've seen the call-by-push-value calculus, its operational and monad/algebra semantics.

The translations from CBV and CBN into CBPV preserve these semantics.

Moggi's TA is UFA.

But

- the monad/algebra semantics makes `thunk` and `force` invisible
- we still don't understand why a function is a "computation".

Antecedents: CBV

- Landin's ISWIM: CBV λ-calculus with effects.
  Influenced ML and other languages.
- Plotkin: λ_{v}-calculus provided equations for CBV with divergence.
- Numerous researchers: CPS transforms for CBV.
- Felleisen *et al*: CBV semantics for various effects.
- Moggi: λ_{c}-calculus, monads for CBV and monadic metalanguage.
- Power and Robinson: Freyd categories for CBV.
- Benton and Kennedy: MIL-lite.
- Thielecke: `thunk` and `force` constructs in CBV with `callcc`.

Antecedents: CBN without η-law for functions

- Plotkin: CPS transform for CBN without η-law.
- Abramsky and Ong: untyped lazy λ-calculus.
- Ong: typed lazy λ-calculus.
- Moggi: monads for CBN without η-law.

Antecedents: CBN with η-law for functions

- Plotkin's PCF, a CBN calculus for recursion.
- Hennessy and Ashcroft: recursion and nondeterminism.
- O'Hearn: semantics of conditional in Reynolds' Idealized Algol.
- Streicher and Reus: semantics of control effects in CBN.
- $\neg\neg$ translations of classical logic. Also CBV.
- Game semantics. Also CBV.

Calculi combining CBV and CBN

- Various calculi based on CPS.
- Filinski's Effect-PCF provided the CBPV `to` construct. U is implicit.
- Howard's $\lambda^{\mu\nu\perp}$ calculus for recursion. U is implicit.
- Egger, Møgelberg and Simpson's Effect Calculus emphasizes connection to Benton's Linear/Nonlinear Logic. U is implicit.
- Marz' SFPL for recursion and sequentiality. F is implicit.
- Nygaard and Winskel's HOPLA for recursion and nondeterminism. F is implicit.
- Laurent's LLP has extra type constructors not included in CBPV.
- Harper, Licata and Zeilberger's Polarized Intuitionistic Logic.

CK-machine

An operational semantics due to Felleisen and Friedman (1986).
And Landin, Krivine, Streicher and Reus, Bierman, Pitts, . . .

It is suitable for sequential languages whether CBV, CBN or CBPV.

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

Initially, K is empty.

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

Transitions for sequencing

To evaluate M `to x.` N: evaluate M. If it returns `return` V, then evaluate $N[V/\mathtt{x}]$.

M `to x.` N	K	$\leadsto$
M	`to x.` $N :: K$	

`return` V	`to x.` $N :: K$	$\leadsto$
$N[V/\mathtt{x}]$	K	

Transitions for application

To evaluate $V`M$: evaluate M. If it returns $\lambda\mathtt{x}.N$, evaluate $N[V/\mathtt{x}]$.

$V`M$	K	$\leadsto$
M	$V :: K$	

$\lambda\mathtt{x}.N$	$V :: K$	$\leadsto$
$N[V/\mathtt{x}]$	K	

Those function rules again

$V`M$	K	$\leadsto$
M	$V :: K$	

$\lambda \mathtt{x}.N$	$V :: K$	$\leadsto$
$N[V/\mathtt{x}]$	K	

Those function rules again

$$\boxed{\begin{array}{lcr} V`M & K & \leadsto \\ M & V :: K & \end{array}}$$

$$\boxed{\begin{array}{lcr} \lambda\mathtt{x}.N & V :: K & \leadsto \\ N[V/\mathtt{x}] & K & \end{array}}$$

We can read $V`$ as an instruction "push V".

We can read $\lambda\mathtt{x}$ as an instruction "pop $\mathtt{x}$".

Those function rules again

$V\text{`}M$	K	$\leadsto$
M	$V :: K$	

$\lambda \mathtt{x}.N$	$V :: K$	$\leadsto$
$N[V/\mathtt{x}]$	K	

We can read $V\text{`}$ as an instruction "push V".

We can read $\lambda\mathtt{x}$ as an instruction "pop $\mathtt{x}$".

Revisiting some equations:

$$\begin{array}{rcll} V \text{ ` } \lambda\mathtt{x}.\ M & = & M[V/\mathtt{x}] & \\ M & = & \lambda\mathtt{x}.\ \mathtt{x} \text{ ` } M & (\mathtt{x}\text{ fresh}) \\ \lambda\mathtt{x}.\ \mathtt{error}\ e & = & \mathtt{error}\ e & \\ \lambda\mathtt{x}.\ \mathtt{print}\ c.\ M & = & \mathtt{print}\ c.\ \lambda\mathtt{x}.M & \end{array}$$

Values and Computations

A value is, a computation does.

- A value of type $U\underline{B}$ is a thunk of a computation of type $\underline{B}$.
- A value of type $\sum_{i \in I} A_i$ is a pair $\langle i, V \rangle$.
- A value of type $A \times A'$ is a pair $\langle V, V' \rangle$.

- A computation of type FA returns a value of type A.
- A computation of type $A \to \underline{B}$
  pops a value of type A
  then behaves in $\underline{B}$.
- A computation of type $\prod_{i \in I} \underline{B}_i$
  pops a tag $i \in I$
  then behaves in $\underline{B}_i$.

What's in a stack?

A stack consists of

- arguments that are values
- arguments that are tags
- frames taking the form $\texttt{to x. } N$.

Example program of type 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".
( print "hello3".
  7‘
  print "we just pushed 7".
  force y
) to w.
print "w is bound to " + w.
return w + 5
```

Typing the CK-machine

Typing the CK-machine

Initial configuration to evaluate $\Gamma \vdash^{\mathsf{c}} P \colon \underline{C}$

Γ	P	$\underline{C}$	$\mathtt{nil}$	$\underline{C}$

Transitions

Γ	$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\underline{B}$	K	$\underline{C}$	$\leadsto$
Γ	M	FA	$\mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$	

Γ	$\mathtt{return}\ V$	FA	$\mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$	$\leadsto$
Γ	$N[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$	

Typically Γ would be empty and $\underline{C} = F\,\mathtt{bool}$.

Typing the CK-machine

Initial configuration to evaluate $\Gamma \vdash^{\mathsf{c}} P : \underline{C}$

Γ	P	$\underline{C}$	$\texttt{nil}$	$\underline{C}$

Transitions

Γ	M $\texttt{to x.}$ N	$\underline{B}$	K	$\underline{C}$	$\rightsquigarrow$
Γ	M	FA	$\texttt{to x.}$ $N :: K$	$\underline{C}$	

Γ	$\texttt{return}$ V	FA	$\texttt{to x.}$ $N :: K$	$\underline{C}$	$\rightsquigarrow$
Γ	$N[V/\texttt{x}]$	$\underline{B}$	K	$\underline{C}$	

Typically Γ would be empty and $\underline{C} = F\,\texttt{bool}$. We write $\Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ to mean that K can accompany a computation of type $\underline{B}$ during evaluation.

Typing rules, read off from the CK-machine

Typing a stack

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

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

$$\frac{\Gamma \mid \underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \mid \prod_{i \in I} \underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}}\ \hat{\imath} \in I$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \mid A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

Typing rules, read off from the CK-machine

Typing a stack

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

$$\frac{\Gamma \mid \underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \mid \prod_{i \in I} \underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \mid A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

Typing a CK-configuration

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \vdash^{\mathsf{ck}} (M, K) : \underline{C}}$$

Special Stacks

Continuations

A continuation is a stack from an F type. For example:

$$\Gamma \mid FA \vdash^{\mathsf{k}} \texttt{to x. } M :: K : \underline{C}$$

It describes what happens next once it receives a value.

Special Stacks

Continuations

A continuation is a stack from an F type. For example:

$$\Gamma \,|\, FA \vdash^{\mathsf{k}} \texttt{to x.}\ M :: K : \underline{C}$$

It describes what happens next once it receives a value.
In CBV, all computations have F type, so all stacks are continuations.

Special Stacks

Continuations

A continuation is a stack from an F type. For example:

$$\Gamma \,|\, FA \vdash^{\mathsf{k}} \mathtt{to}\ \mathtt{x.}\ M :: K : \underline{C}$$

It describes what happens next once it receives a value.
In CBV, all computations have F type, so all stacks are continuations.

Top-Level Stack

The top-level stack is

$$\Gamma \,|\, \underline{C} \vdash^{\mathsf{k}} \mathtt{nil} : \underline{C}$$

and $\underline{C}$ is the top-level type.

Special Stacks

Continuations

A continuation is a stack from an F type. For example:

$$\Gamma \,|\, FA \vdash^{\mathsf{k}} \texttt{to x.}\ M :: K : \underline{C}$$

It describes what happens next once it receives a value.
In CBV, all computations have F type, so all stacks are continuations.

Top-Level Stack

The top-level stack is

$$\Gamma \,|\, \underline{C} \vdash^{\mathsf{k}} \texttt{nil} : \underline{C}$$

and $\underline{C}$ is the top-level type.
If $\underline{C}$ is an F type, then `nil` is the top-level continuation: it receives a value and returns it to the user.

Denotational semantics of stacks

Suppose $[\![\underline{C}]\!] = (Y, \phi)$. The behaviour of $\Gamma \vdash^{\mathsf{ck}} (M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(M,K)]\!]} Y$$

to be preserved by each transition.

Denotational semantics of stacks

Suppose $[\![\underline{C}]\!] = (Y, \phi)$. The behaviour of $\Gamma \vdash^{\mathsf{ck}} (M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(M,K)]\!]} Y$$

to be preserved by each transition.

Suppose also $[\![\underline{B}]\!] = (X, \theta)$. A stack $\Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ transforms computations to CK-configurations. So we get a function

$$[\![\Gamma]\!] \times X \xrightarrow{[\![K]\!]} Y$$

homomorphic in its second argument.

Denotational semantics of stacks

Suppose $[\![\underline{C}]\!] = (Y, \phi)$. The behaviour of $\Gamma \vdash^{\mathsf{ck}} (M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(M,K)]\!]} Y$$

to be preserved by each transition.

Suppose also $[\![\underline{B}]\!] = (X, \theta)$. A stack $\Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ transforms computations to CK-configurations. So we get a function

$$[\![\Gamma]\!] \times X \xrightarrow{[\![K]\!]} Y$$

homomorphic in its second argument.

because if M raises an error or prints, then so does M, K.

Denotational semantics of stacks

Suppose $[\![\underline{C}]\!] = (Y, \phi)$. The behaviour of $\Gamma \vdash^{\mathsf{ck}} (M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(M,K)]\!]} Y$$

to be preserved by each transition.

Suppose also $[\![\underline{B}]\!] = (X, \theta)$. A stack $\Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ transforms computations to CK-configurations. So we get a function

$$[\![\Gamma]\!] \times X \xrightarrow{[\![K]\!]} Y$$

homomorphic in its second argument.

because if M raises an error or prints, then so does M, K.

We assume there's no exception handling.

Adjunction between values and stacks

We have an adjunction between the category of values (sets and functions) and the category of stacks (T-algebras and homomorphisms).

$$
\begin{tikzcd}
\mathbf{Set} \arrow[r, shift left, "F^T"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^T \arrow[l, shift left, "U^T"]
\end{tikzcd}
$$

This resolves the monad T on $\mathbf{Set}$.

State

Consider CBPV extended with two storage cells:
`l` stores a natural number, and `l`$'$ stores a boolean.

State

Consider CBPV extended with two storage cells:
$\mathtt{l}$ stores a natural number, and $\mathtt{l}'$ stores a boolean.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{l} := n.\ M : \underline{B}}\ n \in \mathbb{N} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M_n : \underline{B}\ \ (\forall n \in \mathbb{N})}{\Gamma \vdash^{\mathsf{c}} \mathtt{read\ l\ as}\ \{n.\ M_n\}_{n \in \mathbb{N}} : \underline{B}}$$

State

Consider CBPV extended with two storage cells:
$\texttt{l}$ stores a natural number, and $\texttt{l}'$ stores a boolean.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \texttt{l} := n.\ M : \underline{B}}\ n \in \mathbb{N} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M_n : \underline{B}\ \ (\forall n \in \mathbb{N})}{\Gamma \vdash^{\mathsf{c}} \texttt{read l as}\ \{n.\ M_n\}_{n \in \mathbb{N}} : \underline{B}}$$

A state is $\texttt{l} \mapsto n, \texttt{l}' \mapsto b$.

The set of states is $S \cong \mathbb{N} \times \mathbb{B}$.

Big-step semantics for state

The big-step semantics takes the form $s, M \Downarrow s', T$.

A pair s, M is called an SC-configuration.

We can type these using

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{sc}} (s, M) : \underline{B}} \; s \in S$$

Monad/algebra semantics for state

Moggi's monad for global state is $S \to (S \times -)$.

We can take algebras for this and obtain a denotational semantics of CBPV with state.

Monad/algebra semantics for state

Moggi's monad for global state is $S \to (S \times -)$.

We can take algebras for this and obtain a denotational semantics of CBPV with state.

But it doesn't fit well with SC-configurations.

We'd like a soundness result of the following form:

$$\text{If } s, M \Downarrow s', T \text{ then } [\![s, M]\!]\varepsilon = [\![s', T]\!]\varepsilon$$

This requires an SC-configuration to have a denotation.

Semantics of SC-configurations

Value type A denotes the set of denotations of values of type A. Like in monad semantics.

Semantics of SC-configurations

Value type A denotes the set of denotations of values of type A. Like in monad semantics.

Computation type $[\![\underline{B}]\!]$ denotes the set of behaviours of configurations of type $\underline{B}$.

Semantics of SC-configurations

Value type A denotes the set of denotations of values of type A. Like in monad semantics.

Computation type $[\![\underline{B}]\!]$ denotes the set of behaviours of configurations of type $\underline{B}$.

The behaviour of an SC-configuration $\Gamma \vdash^{\mathsf{sc}} (s, M) : \underline{B}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(s,M)]\!]} [\![\underline{B}]\!]$$

Semantics of SC-configurations

Value type A denotes the set of denotations of values of type A. Like in monad semantics.

Computation type $[\![\underline{B}]\!]$ denotes the set of behaviours of configurations of type $\underline{B}$.

The behaviour of an SC-configuration $\Gamma \vdash^{\mathsf{sc}} (s, M) : \underline{B}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(s,M)]\!]} [\![\underline{B}]\!]$$

The behaviour of a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ depends on the state and environment:

$$S \times [\![\Gamma]\!] \xrightarrow{[\![M]\!]} [\![\underline{B}]\!]$$

An SC-configuration of type FA will terminate as $s, \texttt{return}\ V$.

$$[\![FA]\!] = S \times [\![A]\!]$$

An SC-configuration of type $A \to \underline{B}$ will pop $\texttt{x} : A$, then behave in $\underline{B}$.

$$[\![A \to \underline{B}]\!] = [\![A]\!] \to [\![\underline{B}]\!]$$

An SC-configuration of type $\prod_{i \in I} \underline{B}_i$ will pop $i \in I$, then behave in $\underline{B}_i$.

$$[\![\prod_{i \in I} \underline{B}_i]\!] = \prod_{i \in I} [\![\underline{B}_i]\!]$$

A value $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$ can be forced in any state s, giving an SC-configuration $s, \texttt{force}\ V$.

$$[\![U\underline{B}]\!] = S \to [\![\underline{B}]\!]$$

State: semantics of types

An SC-configuration of type FA will terminate as $s, \mathtt{return}\ V$.

$$[\![FA]\!] = S \times [\![A]\!]$$

An SC-configuration of type $A \to \underline{B}$ will pop $\mathtt{x} : A$, then behave in $\underline{B}$.

$$[\![A \to \underline{B}]\!] = [\![A]\!] \to [\![\underline{B}]\!]$$

An SC-configuration of type $\prod_{i \in I} \underline{B}_i$ will pop $i \in I$, then behave in $\underline{B}_i$.

$$[\![\textstyle\prod_{i \in I} \underline{B}_i]\!] = \textstyle\prod_{i \in I} [\![\underline{B}_i]\!]$$

A value $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$ can be forced in any state s, giving an SC-configuration $s, \mathtt{force}\ V$.

$$[\![U\underline{B}]\!] = S \to [\![\underline{B}]\!]$$

We recover standard semantics for CBV,
and O'Hearn's semantics for CBN.

The SCK-machine

We replace $\vdash^{\mathsf{ck}}$ with $\vdash^{\mathsf{sck}}$.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \vdash^{\mathsf{sck}} (s, M, K) : \underline{C}}\ s \in S$$

The SCK-machine

We replace $\vdash^{\mathsf{ck}}$ with $\vdash^{\mathsf{sck}}$.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \vdash^{\mathsf{sck}} (s, M, K) : \underline{C}}\ s \in S$$

The behaviour of an SCK-configuration $\Gamma \vdash^{\mathsf{sck}} (s, M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(s,M,K)]\!]} [\![\underline{C}]\!]$$

to be preserved by each transition.

The SCK-machine

We replace $\vdash^{\mathsf{ck}}$ with $\vdash^{\mathsf{sck}}$.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \vdash^{\mathsf{sck}} (s, M, K) : \underline{C}}\ s \in S$$

The behaviour of an SCK-configuration $\Gamma \vdash^{\mathsf{sck}} (s, M, K) : \underline{C}$ depends on the environment:

$$[\![\Gamma]\!] \xrightarrow{[\![(s,M,K)]\!]} [\![\underline{C}]\!]$$

to be preserved by each transition.

A stack $\Gamma \,|\, \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ transforms SC-configuration behaviours to SCK-configuration behaviours:

$$[\![\Gamma]\!] \times [\![\underline{B}]\!] \xrightarrow{[\![K]\!]} [\![\underline{C}]\!]$$

State: the value/stack adjunction

We've seen that a stack $\mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ denotes a function $[\![\underline{B}]\!] \xrightarrow{[\![K]\!]} [\![\underline{C}]\!]$.

State: the value/stack adjunction

We've seen that a stack $\mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ denotes a function $[\![\underline{B}]\!] \xrightarrow{[\![K]\!]} [\![\underline{C}]\!]$.

We have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "S\times -"] \arrow[r, phantom, "\perp"] & \mathbf{Set} \arrow[l, shift left, "S\to -"] \end{tikzcd}$$

between values and stacks.

Catching and throwing a stack

We extend CBPV with Crolard's instructions for changing the stack.

- catch α means "let α be the current stack".
- throw K means "change the current stack to K".

Γ	$\texttt{catch}\ \alpha.\ M$	$\underline{B}$	K	Δ	$\leadsto$
Γ	$M[K/\alpha]$	$\underline{B}$	K	Δ	

Γ	$\texttt{throw}\ K.\ M$	$\underline{B}'$	K'	Δ	$\leadsto$
Γ	M	$\underline{B}$	K	Δ	

Typing judgements for control

The stack context Δ consists of stack names $\alpha : \underline{B}$.

$\vdash^{\mathsf{n}}$ indicates "no top-level type".

value	$\Gamma \vdash^{\mathsf{v}} V : A \mid \Delta$	
computation	$\Gamma \vdash^{\mathsf{c}} M : \underline{B} \mid \Delta$	
stack	$\Gamma \mid \underline{B} \vdash^{\mathsf{nk}} K \mid \Delta$	$\Gamma \mid \underline{B} \vdash^{\mathsf{k}} \alpha.\, K : \underline{C} \mid \Delta$
CK-configuration	$\Gamma \vdash^{\mathsf{nck}} (M, K) \mid \Delta$	$\Gamma \vdash^{\mathsf{ck}} \alpha.\, (M, K) : \underline{C} \mid \Delta$

Creating `nil` at start of evaluation

Initial configuration to evaluate $\Gamma \vdash^{\mathsf{c}} P \colon \underline{C} \mid \Delta$

Γ	P	$\underline{C}$	`nil`	$\Delta, \mathtt{nil} : \underline{C}$

Typically Γ and Δ would be empty and $\underline{C} = F\,\mathtt{bool}$.

Monad/algebra semantics of control

Fix a set R, the set of behaviours of CK-configurations.

Monad/algebra semantics of control

Fix a set R, the set of behaviours of CK-configurations.

Moggi's monad for control operators ("continuations") is $(- \to R) \to R$.

Monad/algebra semantics of control

Fix a set R, the set of behaviours of CK-configurations.

Moggi's monad for control operators ("continuations") is $(- \to R) \to R$.

Maybe we can use algebras for this to build a denotational semantics of control.

Semantics of control using stacks

Informally a computation type $[\![\underline{B}]\!]$ denotes the set of stacks from $\underline{B}$.

Semantics of control using stacks

Informally a computation type $[\![\underline{B}]\!]$ denotes the set of stacks from $\underline{B}$.

The behaviour of a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B} \,|\, \Delta$ depends on the environment, stack environment and current stack:

$$[\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\underline{B}]\!] \xrightarrow{[\![M]\!]} R$$

Semantics of control using stacks

Informally a computation type $[\![\underline{B}]\!]$ denotes the set of stacks from $\underline{B}$.

The behaviour of a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B} \mid \Delta$ depends on the environment, stack environment and current stack:

$$[\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\underline{B}]\!] \xrightarrow{[\![M]\!]} R$$

A value $\Gamma \vdash^{\mathsf{v}} V : A \mid \Delta$ denotes

$$[\![\Gamma]\!] \times [\![\Delta]\!] \xrightarrow{[\![V]\!]} [\![A]\!]$$

A stack $\Gamma \mid \underline{B} \vdash^{\mathsf{nk}} K \mid \Delta$ denotes

$$[\![\Gamma]\!] \times [\![\Delta]\!] \xrightarrow{[\![K]\!]} [\![\underline{B}]\!]$$

A CK-configuration $\Gamma \vdash^{\mathsf{nck}} \mathtt{nil}.(M, K) : \Delta$ denotes

$$[\![\Gamma]\!] \times [\![\Delta]\!] \xrightarrow{[\![(M,K)]\!]} R$$

to be preserved by each transition.

Control: semantics of types

A stack from FA receives a value $\mathtt{x} : A$ and then behaves as a configuration.

$$[\![FA]\!] = [\![A]\!] \to R$$

A stack from $A \to \underline{B}$ is a pair $V :: K$.

$$[\![A \to \underline{B}]\!] = [\![A]\!] \times [\![\underline{B}]\!]$$

A stack from $\prod_{i \in I} \underline{B}_i$ is a pair $i :: K$.

$$[\![\prod_{i \in I} \underline{B}_i]\!] = \sum_{i \in I} [\![\underline{B}_i]\!]$$

A value of type $U\underline{B}$ can be forced alongside any stack K, giving a configuration.

$$[\![U\underline{B}]\!] = [\![\underline{B}]\!] \to R$$

Control: semantics of types

A stack from FA receives a value $\mathtt{x} : A$ and then behaves as a configuration.

$$[\![FA]\!] = [\![A]\!] \to R$$

A stack from $A \to \underline{B}$ is a pair $V :: K$.

$$[\![A \to \underline{B}]\!] = [\![A]\!] \times [\![\underline{B}]\!]$$

A stack from $\prod_{i \in I} \underline{B}_i$ is a pair $i :: K$.

$$[\![\prod_{i \in I} \underline{B}_i]\!] = \sum_{i \in I} [\![\underline{B}_i]\!]$$

A value of type $U\underline{B}$ can be forced alongside any stack K, giving a configuration.

$$[\![U\underline{B}]\!] = [\![\underline{B}]\!] \to R$$

We recover standard continuation semantics for CBV, and Streicher and Reus' semantics for CBN.

Control: the value/stack adjunction

A stack with top-level type

$$\Gamma \mid \underline{B} \vdash^{\mathsf{k}} \alpha.\, K : \underline{C} \mid \Delta$$

denotes a function

$$[\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\underline{C}]\!] \xrightarrow{[\![\alpha.\, K]\!]} [\![\underline{B}]\!]$$

Control: the value/stack adjunction

A stack with top-level type

$$\Gamma \mid \underline{B} \vdash^{\mathsf{k}} \alpha.\, K : \underline{C} \mid \Delta$$

denotes a function

$$[\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\underline{C}]\!] \xrightarrow{[\![\alpha.\, K]\!]} [\![\underline{B}]\!]$$

So we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "- \to R"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^{\mathsf{op}} \arrow[l, shift left, "- \to R"] \end{tikzcd}$$

between values and stacks with top-level type.

Summary of models

For every monad T on $\mathbf{Set}$ we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "F^T"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^T \arrow[l, shift left, "U^T"] \end{tikzcd}$$

This is useful for modelling CBPV with errors and printing.

Summary of models

For every monad T on $\mathbf{Set}$ we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "F^T"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^T \arrow[l, shift left, "U^T"] \end{tikzcd}$$

This is useful for modelling CBPV with errors and printing.

For a set S we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "S\times -"] \arrow[r, phantom, "\perp"] & \mathbf{Set} \arrow[l, shift left, "S\to -"] \end{tikzcd}$$

This is useful for modelling CBPV with state.

Summary of models

For every monad T on $\mathbf{Set}$ we have an adjunction

$$\mathbf{Set} \underset{U^T}{\overset{F^T}{\rightleftarrows}} \mathbf{Set}^T \quad (\perp)$$

This is useful for modelling CBPV with errors and printing.

For a set S we have an adjunction

$$\mathbf{Set} \underset{S\to -}{\overset{S\times -}{\rightleftarrows}} \mathbf{Set} \quad (\perp)$$

This is useful for modelling CBPV with state.

For a set R we have an adjunction

$$\mathbf{Set} \underset{-\to R}{\overset{-\to R}{\rightleftarrows}} \mathbf{Set}^{\mathsf{op}} \quad (\perp)$$

This is useful for modelling CBPV with control.

Summary of models

For every monad T on $\mathbf{Set}$ we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "F^T"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^T \arrow[l, shift left, "U^T"] \end{tikzcd}$$

This is useful for modelling CBPV with errors and printing.

For a set S we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "S\times -"] \arrow[r, phantom, "\perp"] & \mathbf{Set} \arrow[l, shift left, "S\to -"] \end{tikzcd}$$

This is useful for modelling CBPV with state.

For a set R we have an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "-\to R"] \arrow[r, phantom, "\perp"] & \mathbf{Set}^{\mathsf{op}} \arrow[l, shift left, "-\to R"] \end{tikzcd}$$

This is useful for modelling CBPV with control.