A calculus of expandable stores

Continuation-and-environment-passing style translations

Hugo Herbelin[1] Étienne Miquey[2]

[1]Inria
Université de Paris, CNRS, IRIF

[2]CNRS
ÉNS Paris-Saclay, Inria, LSV

LICS 2020

A computational wonderland

The λ-calculus

One calculus to rule them all

A very nice abstraction:

- Turing-complete
- different evaluation strategies
- different type systems
- pure and effectful computations

Operational semantics through **abstract machines**

↪ *SECD (Landin), KAM (Krivine), CEK (Felleisen and Friedman), ZINC (Leroy)...*

Continuation-passing style (CPS) translations allow to abstract the machine again.

- specify an evaluation strategy
- make explicit the control flow
- induce a type translation ≡ **syntactic model**

  ↪ *allowing to transfer logical properties from the target calculus*

A computational wonderland

The λ-calculus

One calculus to rule them all

A very nice abstraction:

- Turing-complete
- different type systems
- different evaluation strategies
- pure and effectful computations

Operational semantics through **abstract machines**

$\hookrightarrow$ *SECD (Landin), KAM (Krivine), CEK (Felleisen and Friedman), ZINC (Leroy)...*

Continuation-passing style (CPS) translations allow to abstract the machine again.

- specify an evaluation strategy
- make explicit the control flow
- induce a type translation $\equiv$ **syntactic model**
  - $\hookrightarrow$ *allowing to transfer logical properties from the target calculus*

A computational wonderland

The λ-calculus

One calculus to rule them all

A very nice abstraction:

- Turing-complete
- different type systems
- different evaluation strategies
- pure and effectful computations

Operational semantics through **abstract machines**

↬ *SECD (Landin), KAM (Krivine), CEK (Felleisen and Friedman), ZINC (Leroy)...*

Continuation-passing style (CPS) translations allow to abstract the machine again.

- specify an evaluation strategy
- make explicit the control flow
- induce a type translation ≡ **syntactic model**

↬ *allowing to transfer logical properties from the target calculus*

A computational wonderland

The λ-calculus

One calculus to rule them all

A very nice abstraction:

- Turing-complete
- different type systems
- different evaluation strategies
- pure and effectful computations

Operational semantics through **abstract machines**

$\looparrowright$ *SECD (Landin), KAM (Krivine), CEK (Felleisen and Friedman), ZINC (Leroy)...*

Continuation-passing style (CPS) translations allow to abstract the machine again.

- specify an evaluation strategy
- make explicit the control flow
- induce a type translation ≡ **syntactic model**

  $\looparrowright$ *allowing to transfer logical properties from the target calculus*

In praise of laziness

Call-by-need evaluation strategy:

- evaluates arguments of functions only when needed
  ↬ *as in call-by-name*
- shares the evaluations across all places where they are needed
  ↬ *as in call-by-value*

In short:

demand-driven computations + **memoization**

Many benefits, used in **Haskell** (by default) or **Coq** (tactic, kernel).

Trickier and historically less studied than CbName/CbValue.

In praise of laziness

Call-by-need evaluation strategy:

- evaluates arguments of functions only when needed
  ↬ *as in call-by-name*
- shares the evaluations across all places where they are needed
  ↬ *as in call-by-value*

In short:

demand-driven computations + **memoization**

Many benefits, used in **Haskell** (by default) or **Coq** (tactic, kernel).

Trickier and historically less studied than CbName/CbValue.

In praise of laziness

Call-by-need evaluation strategy:

- evaluates arguments of functions only when needed
  ↬ *as in call-by-name*
- shares the evaluations across all places where they are needed
  ↬ *as in call-by-value*

In short:

demand-driven computations + **memoization**

Many benefits, used in **Haskell** (by default) or **Coq** (tactic, kernel).

Trickier and historically less studied than CbName/CbValue.

Computing with global environments

Standard abstract machines use local environments and closures:

Krivine Abstract Machine (CbName)

$$
\begin{array}{lllll}
t\,u \star S \star E & \rightarrow_c & t \star (u, E) \cdot S \star E \\
\lambda x.t \star (u, E') \cdot S \star E & \rightarrow_\beta & t \star S \star E[x ::= (u, E')] \\
x \star S \star E[x ::= (t, E')]E'' & \rightarrow_s & t \star S \star E'
\end{array}
$$

Call-by-need requires a global environment to share computations.

Milner Abstract Machine (CbName)

$$
\begin{array}{lll}
tu \star \pi \star \tau & \rightarrow_c & t \star u \cdot \pi \star \tau \\
\lambda x.t \star u \cdot \pi \star \tau & \rightarrow_\beta & t \star \pi \star \tau[x := u] \\
x \star \pi \star \tau[x := t]\tau' & \rightarrow_s & \bar{t}^\alpha \star \pi \star \tau[x := t]\tau'
\end{array}
$$

Globality requires to explicitly handle addresses or a **renaming process**.

Computing with global environments

Standard abstract machines use local environments and closures:

Krivine Abstract Machine (CbName)

$$\begin{array}{lllll}
t\,u \star S \star E & \rightarrow_{\mathsf{c}} & t \star (u, E) \cdot S \star E \\
\lambda x.t \star (u, E') \cdot S \star E & \rightarrow_{\beta} & t \star S \star E[x ::= (u, E')] \\
x \star S \star E[x ::= (t, E')]E'' & \rightarrow_{\mathsf{s}} & t \star S \star E'
\end{array}$$

Call-by-need requires a global environment to share computations.

Milner Abstract Machine (CbName)

$$\begin{array}{lll}
tu \star \pi \star \tau & \rightarrow_{\mathsf{c}} & t \star u \cdot \pi \star \tau \\
\lambda x.t \star u \cdot \pi \star \tau & \rightarrow_{\beta} & t \star \pi \star \tau[x := u] \\
x \star \pi \star \tau[x := t]\tau' & \rightarrow_{\mathsf{s}} & \bar{t}^{\alpha} \star \pi \star \tau[x := t]\tau'
\end{array}$$

Globality requires to explicitly handle addresses or a **renaming process**.

Computing with global environments

Standard abstract machines use local environments and closures:

Krivine Abstract Machine (CbName)

$$\begin{array}{lclcl}
t\,u \star S \star E & \to_c & t \star (u, E) \cdot S \star E \\
\lambda x.t \star (u, E') \cdot S \star E & \to_\beta & t \star S \star E[x ::= (u, E')] \\
x \star S \star E[x ::= (t, E')]E'' & \to_s & t \star S \star E'
\end{array}$$

Call-by-need requires a global environment to share computations.

Milner Abstract Machine (CbName)

$$\begin{array}{lcl}
tu \star \pi \star \tau & \to_c & t \star u \cdot \pi \star \tau \\
\lambda x.t \star u \cdot \pi \star \tau & \to_\beta & t \star \pi \star \tau[x := u] \\
x \star \pi \star \tau[x := t]\tau' & \to_s & \bar{t}^{\alpha} \star \pi \star \tau[x := t]\tau'
\end{array}$$

Globality requires to explicitly handle addresses or a **renaming process**.

A thorn in the side

A lost paradise?

- ✓ Abstract machines with global environments
- ✓ By-need abstract machines
  - ↬ *Sestoft's machine, Accattoli, Barenbaum and Mazza's Merged MAD*
- ✗ Typed **continuation-and-environment passing** style translation?

Several difficulties to handle:

- How should control and environments interact?
- Can we soundly type environments?
- ... while accounting for extensibility?
- How to avoid name clashes?

This paper

Our goal

Typed continuation-and-environment-passing style (CEPS) translations

↬ *i.e. understand how to soundly CEPS translate calculi with global environments*

Contribution

- We introduce F_Υ, a **generic** calculus used as the target of CEPS translations, which features:
  - a data type for **typed stores**
  - **explicit coercions** witnessing store extensions

- We use it to implement simply-typed CEPS translations for:
  ✓ call-by-need ✓ call-by-name ✓ call-by-value

This paper

Our goal

Typed continuation-and-environment-passing style (CEPS) translations

$\looparrowright$ *i.e. understand how to soundly CEPS translate calculi with global environments*

Contribution

- We introduce F_Υ, a **generic** calculus used as the target of CEPS translations, which features:
  - a data type for **typed stores**
  - **explicit coercions** witnessing store extensions

- We use it to implement simply-typed CEPS translations for:
  ✓ call-by-need ✓ call-by-name ✓ call-by-value

This paper

Our goal

Typed continuation-and-environment-passing style (CEPS) translations

↬ *i.e. understand how to soundly CEPS translate calculi with global environments*

Contribution

- We introduce F_Υ, a **generic** calculus used as the target of CEPS translations, which features:
  - a data type for **typed stores**
  - **explicit coercions** witnessing store extensions

Generic?

We aim at isolating the key ingredients necessary to the definition of well-typed CEPS translations.

This paper

Our goal

Typed continuation-and-environment-passing style (CEPS) translations

$\looparrowright$ *i.e. understand how to soundly CEPS translate calculi with global environments*

Contribution

- We introduce F_Υ, a **generic** calculus used as the target of CEPS translations, which features:
  - a data type for **typed stores**
  - **explicit coercions** witnessing store extensions
- We use it to implement simply-typed CEPS translations for:

  ✓ call-by-need ✓ call-by-name ✓ call-by-value

Continuation-and-environment passing style translations

Towards typed translations

Backtrack and laziness

Question

What should be the semantics of a control operator in presence of a shared memory?

```
let a = catch_k (fun k ⇒
  (Id ,fun x ⇒ throw k x))
    f = fst a
    q = snd a
in f q (Id , Id)
```

Okasaki, Lee & Tarditi '93:

What does not force the effect is shared.

- q shared
- f recomputed

↬ *loops...*

Backtrack and laziness

Question

What should be the semantics of a control operator in presence of a shared memory?

```
let a = catch_k (fun k ⇒
  (Id , fun x ⇒ throw k x))
    f = fst a
    q = snd a
in f q (Id , Id)
```

Okasaki, Lee & Tarditi '93:

What does not force the effect is shared.

- q shared
- f recomputed

↬ *loops...*

Ariola *et al.* '12:

Nothing is shared inside an effect

- f recomputed
- q recomputed

↬ *returns (Id,Id)* ✓

Method:

1. sequent calculus
2. abstract machine
3. (untyped) CPS translation
4. realizability interpretation

Backtrack and laziness

Theorem [M.-Herbelin '18]

Ariola *et al.*'s semantics is typable, normalizing and consistent.

```
let a = catch_k (fun k ⇒
  (Id, fun x ⇒ throw k x))
    f = fst a
    q = snd a
in f q (Id, Id)
```

Okasaki, Lee & Tarditi '93:

What does not force the effect is shared.

- q shared
- f recomputed

↬ *loops...*

Ariola *et al.* '12:

Nothing is shared inside an effect

- f recomputed
- q recomputed

↬ *returns (Id,Id)* ✓

Method:

1. sequent calculus
2. abstract machine
3. (untyped) CPS translation
4. realizability interpretation

Intuitions

(Analyzing Ariola *et al.* '12)

Sequent calculus:

$$\langle t \parallel e \rangle \tau$$

$$\nearrow \qquad \uparrow \qquad \nwarrow$$

Term Context Environment

Syntax

Terms

Terms	$t, u ::= V \mid \mu\alpha.c$
Weak val.	$V ::= v \mid x$
Strong val.	$v ::= \lambda x.t \mid \boldsymbol{k}$

Contexts

Contexts	$e ::= E \mid \tilde{\mu}x.c$
Catchable cont.	$E ::= F \mid \alpha \mid \tilde{\mu}[x].\langle x \parallel F\rangle\tau$
Forcing cont.	$F ::= t \cdot E \mid \boldsymbol{\kappa}$

Environments	$\tau ::= \varepsilon \mid \tau[x := t] \mid \tau[\alpha := E]$
Commands	$c ::= \langle t \parallel e \rangle$

Intuitions

(Analyzing Ariola *et al.* '12)

Sequent calculus:

$$\langle t \parallel e \rangle \tau$$

$$\nearrow \qquad \uparrow \qquad \nwarrow$$

Term Context Environment

Syntax

Terms

Terms	$t, u ::= V \mid \mu\alpha.c$
Weak val.	$V ::= v \mid x$
Strong val.	$v ::= \lambda x.t \mid \boldsymbol{k}$

Contexts

Contexts	$e ::= E \mid \tilde{\mu}x.c$
Catchable cont.	$E ::= F \mid \alpha \mid \tilde{\mu}[x].\langle x \parallel F\rangle\tau$
Forcing cont.	$F ::= t \cdot E \mid \boldsymbol{\kappa}$

Environments	$\tau ::= \varepsilon \mid \tau[x := t] \mid \tau[\alpha := E]$
Commands	$c ::= \langle t \parallel e \rangle$

Intuitions

(Analyzing Ariola *et al.* ’12)

Syntax

Terms

Terms	$t, u ::= V \mid \mu\alpha.c$
Weak val.	$V ::= v \mid x$
Strong val.	$v ::= \lambda x.t \mid \boldsymbol{k}$

Contexts

Contexts	$e ::= E \mid \tilde\mu x.c$
Catchable cont.	$E ::= F \mid \alpha \mid \tilde\mu[x].\langle x \parallel F\rangle\tau$
Forcing cont.	$F ::= t \cdot E \mid \boldsymbol{\kappa}$

Environments	$\tau ::= \varepsilon \mid \tau[x := t] \mid \tau[\alpha := E]$
Commands	$c ::= \langle t \parallel e\rangle$

Lazy reduction:

(Lazy storage)	$\langle t \parallel \tilde\mu x.c\rangle\tau$	$\to$	$c\tau[x := t]$
(Catch)	$\langle \mu\alpha.c \parallel E\rangle\tau$	$\to$	$c\tau[\alpha := E]$
(Lookup)	$\langle x \parallel F\rangle\tau[x := \boldsymbol{t}]\tau'$	$\to$	$\langle \boldsymbol{t} \parallel \tilde\mu[x].\langle x \parallel F\rangle\tau'\rangle\tau$
(Forced eval.)	$\langle \boldsymbol{V} \parallel \tilde\mu[x].\langle x \parallel F\rangle\tau'\rangle\tau$	$\to$	$\langle \boldsymbol{V} \parallel F\rangle\tau[x := \boldsymbol{V}]\tau'$
	$\langle \lambda x.t \parallel u \cdot E\rangle\tau$	$\to$	$\langle u \parallel \tilde\mu x.\langle t \parallel E\rangle\rangle\tau$

Intuitions

(Analyzing Ariola *et al.* '12)

Sequent calculus:

$$\langle t \parallel e \rangle \tau$$

Term Context Environment

Untyped CEPS:

$$[\![\langle t \parallel e \rangle \tau]\!] \quad \simeq \quad [\![e]\!]_{\mathrm{e}}\ [\![\tau]\!]_{\tau}\ [\![t]\!]_{\mathrm{t}}$$

environment passing

continuation passing

Intuitions

(Analyzing Ariola *et al.* '12)

Untyped CEPS:

$$[\![\langle t \parallel e \rangle \tau]\!] \quad \simeq \quad [\![e]\!]_{\mathsf{e}} \; \color{blue}{[\![\tau]\!]_\tau} \; \color{red}{[\![t]\!]_{\mathsf{t}}}$$

environment passing $\nearrow$ $\nwarrow$ continuation passing

$$\begin{array}{rcl}
[\![\tilde\mu x.c]\!]_{\mathsf{e}} & := & \lambda \color{blue}{\tau} \color{red}{t}\color{black}{.[\![c]\!]_{\mathsf{c}}\, \tau[x := t]} \\
[\![E]\!]_{\mathsf{e}} & := & \lambda \color{blue}{\tau} \color{red}{t}\color{black}{.t\, \tau\, [\![E]\!]_{\mathsf{E}}} \\
[\![\mu\alpha.c]\!]_{\mathsf{t}} & := & \lambda \color{blue}{\tau} \color{red}{E}\color{black}{.([\![c]\!]_{\mathsf{c}}\, \tau)[E/\alpha]} \\
[\![V]\!]_{\mathsf{t}} & := & \lambda \color{blue}{\tau} \color{red}{E}\color{black}{.E\, \tau\, [\![V]\!]_{\mathsf{v}}} \\
[\![\tilde\mu[x].\langle x \parallel F \rangle \tau']\!]_{\mathsf{E}} & := & \lambda \color{blue}{\tau} \color{red}{V}\color{black}{.V\, \tau[x := V][\![\tau']\!]_\tau\, [\![F]\!]_{\mathsf{f}}} \\
[\![F]\!]_{\mathsf{E}} & := & \lambda \color{blue}{\tau} \color{red}{V}\color{black}{.V\, \tau\, [\![F]\!]_{\mathsf{f}}} \\
[\![x]\!]_{\mathsf{v}} & := & \lambda \color{blue}{\tau} \color{red}{F}\color{black}{.\tau(x)\, \tau\, (\lambda} \color{blue}{\tau} \color{red}{V}\color{black}{.V\, \tau[x := V]\tau'\, [\![F]\!]_{\mathsf{f}})} \\
[\![\lambda x.t]\!]_{\mathsf{v}} & := & \lambda \color{blue}{\tau} \color{red}{F}\color{black}{.F\, \tau\, (\lambda u \tau E.[\![t]\!]_{\mathsf{t}}\, \tau[x := u]\, E)} \\
[\![u \cdot E]\!]_{\mathsf{f}} & := & \lambda \color{blue}{\tau} \color{red}{v}\color{black}{.v\, [\![t]\!]_{\mathsf{t}}\, \tau\, [\![E]\!]_{\mathsf{E}}}
\end{array}$$

Typing the CEPS: guidelines (1/4)

$$[\![\langle t \,\|\, e\rangle\tau]\!] \quad\simeq\quad [\![e]\!]_{\mathrm{e}}\ [\![\tau]\!]_{\tau}\ [\![t]\!]_{\mathrm{t}}$$

environment passing

continuation passing

Typing the CEPS: guidelines (1/4)

$$[\![\langle t \parallel e\rangle\tau]\!] \quad\simeq\quad [\![e]\!]_{\mathrm{e}}\ \underbrace{[\![\tau]\!]_{\tau}}_{\text{environment passing}}\ \underbrace{[\![t]\!]_{\mathrm{t}}}_{\text{continuation passing}}$$

Step 1 - Continuation-passing part

$$\boxed{\Gamma \vdash_{\mathrm{t}} t : A}$$

$$\downarrow$$

$$\boxed{[\![\Gamma]\!] \vdash [\![t]\!]_{\mathrm{t}} : [\![A]\!]_{\mathrm{t}}}$$

Typing the CEPS: guidelines (1/4)

$$[\![\langle t \parallel e\rangle\tau]\!] \quad\simeq\quad [\![e]\!]_{\mathsf{e}}\ [\![\tau]\!]_{\tau}\ [\![t]\!]_{\mathsf{t}}$$

environment passing — continuation passing

Step 1 - Continuation-passing part

$$\begin{array}{rcl}
[\![A]\!]_{\mathsf{e}} & \triangleq & [\![A]\!]_{\mathsf{t}} \to \bot \\
[\![A]\!]_{\mathsf{t}} & \triangleq & [\![A]\!]_{\mathsf{E}} \to \bot \\
[\![A]\!]_{\mathsf{E}} & \triangleq & [\![A]\!]_{\mathsf{V}} \to \bot \\
[\![A]\!]_{\mathsf{V}} & \triangleq & [\![A]\!]_{\mathsf{F}} \to \bot \\
[\![A]\!]_{\mathsf{F}} & \triangleq & [\![A]\!]_{\mathsf{v}} \to \bot \\
[\![A \to B]\!]_{\mathsf{v}} & \triangleq & [\![A]\!]_{\mathsf{t}} \to [\![B]\!]_{\mathsf{E}} \to \bot
\end{array}
\qquad
\begin{array}{rcl}
[\![\tilde{\mu}x.c]\!]_{\mathsf{e}} & = & \lambda t.[\![c]\!]_{\mathsf{c}} \\
[\![\mu\alpha.c]\!]_{\mathsf{t}} & = & \lambda\alpha.[\![c]\!]_{\mathsf{c}} \\
& \vdots &
\end{array}$$

↬ *In comparison, for call-by-name/call-by-value we would only have 4/3 layers.*

Typing the CEPS: guidelines (2/4)

$$[\![\langle t \parallel e\rangle\tau]\!] \quad \simeq \quad [\![e]\!]_e\ [\![\tau]\!]_\tau\ [\![t]\!]_t$$

environment passing

continuation passing

Step 2- Environment-passing part

$$\boxed{\Gamma \vdash_t t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_t : [\![\Gamma]\!] \rightarrow [\![A]\!]_t}$$

Typing the CEPS: guidelines (2/4)

$$[\![\langle t \parallel e \rangle \tau]\!] \quad \simeq \quad [\![e]\!]_{\mathrm{e}}\ [\![\tau]\!]_{\tau}\ [\![t]\!]_{\mathrm{t}}$$

environment passing

continuation passing

Step 2- Environment-passing part

$$\boxed{\Gamma \vdash_{\mathrm{t}} t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_{\mathrm{t}} : [\![\Gamma]\!] \rhd_{\mathrm{t}} A}$$

Typing the CEPS: guidelines (2/4)

$$[\![\langle t \parallel e\rangle\tau]\!] \quad \simeq \quad [\![e]\!]_e\, \underbrace{[\![\tau]\!]_\tau}_{\text{environment passing}}\, \underbrace{[\![t]\!]_t}_{\text{continuation passing}}$$

Step 2- Environment-passing part

$$\boxed{\Gamma \vdash_t t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_t : [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_E A \to \bot}$$

Typing the CEPS: guidelines (2/4)

$$[\![\langle t \parallel e \rangle \tau]\!] \quad \simeq \quad [\![e]\!]_{\mathrm{e}}\ [\![\tau]\!]_\tau\ [\![t]\!]_{\mathrm{t}}$$

environment passing

continuation passing

Step 2- Environment-passing part

$$\boxed{\Gamma \vdash_{\mathrm{t}} t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_{\mathrm{t}} : [\![\Gamma]\!] \to ([\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{V}} A \to \bot) \to \bot}$$

Typing the CEPS: guidelines (2/4)

$$[\![\langle t \,\|\, e\rangle\tau]\!] \quad\simeq\quad [\![e]\!]_{\mathrm{e}}\ [\![\tau]\!]_{\tau}\ [\![t]\!]_{\mathrm{t}}$$

environment passing

continuation passing

Step 2- Environment-passing part

$$\begin{array}{rcl}
[\![\Gamma]\!] \triangleright_{\mathrm{e}} A & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{t}} A \to \bot \\
[\![\Gamma]\!] \triangleright_{\mathrm{t}} A & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{E}} A \to \bot \\
[\![\Gamma]\!] \triangleright_{\mathrm{E}} A & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{V}} A \to \bot \\
[\![\Gamma]\!] \triangleright_{\mathrm{V}} A & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{F}} A \to \bot \\
[\![\Gamma]\!] \triangleright_{\mathrm{F}} A & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{v}} A \to \bot \\
[\![\Gamma]\!] \triangleright_{\mathrm{v}} A \to B & \triangleq & [\![\Gamma]\!] \to [\![\Gamma]\!] \triangleright_{\mathrm{t}} A \to [\![\Gamma]\!] \triangleright_{\mathrm{E}} B \to \bot
\end{array}$$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

A possible reduction scheme:

t is needed $\langle x \,\|\, F\rangle\tau_1[x := t]\tau_2$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

A possible reduction scheme:

t is needed	$\langle x \,\|\, F\rangle \tau_1 [x := t] \tau_2$
evaluation of t	$\rightarrow \langle t \,\|\, \tilde{\mu}[x].\langle x \,\|\, F\rangle \tau_2 \rangle \tau_1$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

A possible reduction scheme:

t is needed	$\langle x \parallel F\rangle\tau_1[x := t]\tau_2$
evaluation of t	$\to \langle t \parallel \tilde{\mu}[x].\langle x \parallel F\rangle\tau_2\rangle\tau_1$
t produces a value	$\to^* \langle V \parallel \tilde{\mu}[x].\langle x \parallel F\rangle\tau_2\rangle\tau_1 \boxed{\tau'}$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

A possible reduction scheme:

t is needed	$\langle x \parallel F\rangle\tau_1[x := t]\tau_2$
evaluation of t	$\rightarrow \langle t \parallel \tilde{\mu}[x].\langle x \parallel F\rangle\tau_2\rangle\tau_1$
t produces a value	$\rightarrow^* \langle V \parallel \tilde{\mu}[x].\langle x \parallel F\rangle\tau_2\rangle\tau_1 \boxed{\tau'}$
V is stored	$\rightarrow \langle V \parallel F\rangle\tau_1\tau'[x := V]\tau_2$

Key idea:

$[\![t]\!]_t : [\![\Gamma]\!] \triangleright_t A$ should be compatible with any extension of $[\![\Gamma]\!]$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

$[\![t]\!]_{\mathrm{t}} : [\![\Gamma]\!] \triangleright_{\mathrm{t}} A$ should be compatible with any extension of $[\![\Gamma]\!]$

Store subtyping:

$$\boxed{\Gamma' <: \Gamma}$$

$$\boxed{B_1}\boxed{B_2}\boxed{A_1}\boxed{B_3}\boxed{A_2}\boxed{B_4} \quad <: \quad \boxed{A_1}\boxed{A_2}$$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

$[\![t]\!]_t : [\![\Gamma]\!] \triangleright_t A$ should be compatible with any extension of $[\![\Gamma]\!]$

Store subtyping:

$$\boxed{\Gamma' <: \Gamma}$$

Translation:

$$\boxed{\Gamma \vdash_t t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_t : [\![\Gamma]\!] \rightarrow [\![\Gamma]\!] \triangleright_E A \rightarrow \bot}$$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

$[\![t]\!]_{\mathsf{t}} : [\![\Gamma]\!] \triangleright_{\mathsf{t}} A$ should be compatible with any extension of $[\![\Gamma]\!]$

Store subtyping:

$$\boxed{\Gamma' <: \Gamma}$$

Translation:

$$\boxed{\Gamma \vdash_{\mathsf{t}} t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_{\mathsf{t}} : \forall \Upsilon <: [\![\Gamma]\!] . \Upsilon \to \Upsilon \triangleright_{\mathsf{E}} A \to \bot}$$

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

$[\![t]\!]_{\mathrm{t}} : [\![\Gamma]\!] \triangleright_{\mathrm{t}} A$ should be compatible with any extension of $[\![\Gamma]\!]$

Store subtyping:

$$\boxed{\Gamma' <: \Gamma}$$

Translation:

$$\boxed{\Gamma \vdash_{\mathrm{t}} t : A}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_{\mathrm{t}} : \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to (\forall \Upsilon' <: \Upsilon.\Upsilon' \to \Upsilon' \triangleright_V A \to \bot) \to \bot}$$

(reminiscent of Kripke forcing)

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

$[\![t]\!]_t : [\![\Gamma]\!] \triangleright_t A$ should be compatible with any extension of $[\![\Gamma]\!]$

Store subtyping:

$$\boxed{\Gamma' <: \Gamma}$$

Translation:

$$\begin{array}{rcl}
[\![\Gamma]\!] \triangleright_e A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_t A \to \bot \\
[\![\Gamma]\!] \triangleright_t A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_E A \to \bot \\
[\![\Gamma]\!] \triangleright_E A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_V A \to \bot \\
[\![\Gamma]\!] \triangleright_V A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_F A \to \bot \\
[\![\Gamma]\!] \triangleright_F A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_v A \to \bot \\
[\![\Gamma]\!] \triangleright_v A \to B & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_t A \to \Upsilon \triangleright_E B \to \bot
\end{array}$$

Typing the CEPS: guidelines (4/4)

Step 4 - Avoiding name clashes

Ariola *et al.* work implicit relies on α-renaming on-the-fly.

$\looparrowright$ *incompatible with the CEPS translation*

Typing the CEPS: guidelines (4/4)

Step 4 - Avoiding name clashes

Ariola *et al.* work implicit relies on α-renaming on-the-fly.

↬ *incompatible with the CEPS translation*

Here, we use De Bruijn levels both:

- in the source:

$$\frac{\Gamma(n) = (x_n : T)}{\Gamma \vdash_V x_n : T} \qquad \begin{array}{rcl} \langle x_n \parallel F\rangle\tau[x_n := t]\tau & \xrightarrow{n=|\tau|} & \langle t \parallel \tilde{\mu}[x_n].\langle x_n \parallel F\rangle\tau'\rangle\tau \\ \langle V \parallel \tilde{\mu}[x_i].\langle x_i \parallel F\rangle\tau'\rangle\tau & \xrightarrow{n=|\tau|} & \langle V \parallel \uparrow_i^n F\rangle\tau[x_n := V]\uparrow_i^n\tau' \end{array}$$

Typing the CEPS: guidelines (4/4)

Step 4 - Avoiding name clashes

Ariola *et al.* work implicit relies on α-renaming on-the-fly.

$\looparrowright$ *incompatible with the CEPS translation*

Here, we use De Bruijn levels both:

- and the target:

$$\boxed{x_0 : A, \alpha_1 : B^\perp, x_2 : C \vdash_{\mathrm{t}} t : D}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_{\mathrm{t}} : A, B^\perp, C \triangleright_{\mathrm{t}} D}$$

Typing the CEPS: guidelines (4/4)

Step 4 - Avoiding name clashes

Here, we use De Bruijn levels both:

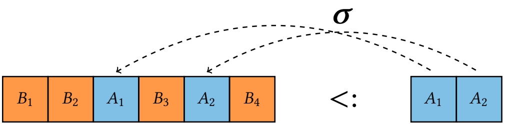
- and the target:

$$\boxed{x_0 : A, \alpha_1 : B^\perp, x_2 : C \vdash_t t : D}$$

$$\downarrow$$

$$\boxed{\vdash [\![t]\!]_t : A, B^\perp, C \triangleright_t D}$$

...where we use **coercions** $\boxed{\sigma : \Gamma' <: \Gamma}$ to witness store extension and keep track of De Bruijn:



A calculus of expandable stores

Introducing F_Υ

Principles

The motto

System F_Υ defines a *parametric* target for CEPS translations

Each CEPS translation can be divided in three blocks:

1. a **source calculus** and its type system
2. a syntax for **stores and coercions**
3. the **target calculus**, an instance of F_Υ

Principles

The motto

System F_Υ defines a *parametric* target for CEPS translations

Each CEPS translation can be divided in three blocks:

1. a **source calculus** and its type system
   $\looparrowright$ *Here, simply-typed calculi*
2. a syntax for **stores and coercions**
3. the **target calculus**, an instance of F_Υ

Principles

The motto

System F_Υ defines a *parametric* target for CEPS translations

Each CEPS translation can be divided in three blocks:

1. a **source calculus** and its type system
2. a syntax for **stores and coercions**
3. the **target calculus**, an instance of F_Υ

Principles

The motto

System F_Υ defines a *parametric* target for CEPS translations

Each CEPS translation can be divided in three blocks:

1. a **source calculus** and its type system
2. a syntax for **stores and coercions**
3. the **target calculus**, an instance of F_Υ

Stores $\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

In this paper, we only use **lists** to represent stores:

$$
\begin{array}{llcl}
\textbf{Source types} & A & ::= & X \mid A \to B \qquad\qquad F ::= A \mid A^\perp \\
\textbf{Store types} & \Upsilon & ::= & Y \mid \emptyset \mid \Upsilon, F \mid \Upsilon;\Upsilon' \\
\textbf{Stores} & \tau & ::= & \delta \mid [] \mid \tau[t] \mid \tau;\tau'
\end{array}
$$

$\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

“Appended to a store of type Υ', the store τ is of type Υ.”

$$
\frac{}{\Gamma \vdash [] : \emptyset \triangleright_\tau \emptyset}
\qquad
\frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T}
\qquad
\frac{\Gamma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon \quad \Gamma \vdash \tau' : (\Upsilon_0;\Upsilon) \triangleright_\tau \Upsilon'}{\Gamma \vdash \tau;\tau' : \Upsilon_0 \triangleright_\tau \Upsilon;\Upsilon'}
$$

Remark

type of a store = *list of source types*

how *these types are translated* = $\blacktriangleright$ = ***parameter*** *of the target*

Stores

$\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

In this paper, we only use **lists** to represent stores:

Source types	A	$::=$	$X \mid A \to B$	$F ::= A \mid A^\perp$
Store types	Υ	$::=$	$Y \mid \emptyset \mid \Upsilon, F \mid \Upsilon;\Upsilon'$	
Stores	τ	$::=$	$\delta \mid [] \mid \tau[t] \mid \tau;\tau'$	

$\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

"Appended to a store of type Υ', the store τ is of type Υ."

$$\frac{}{\Gamma \vdash [] : \emptyset \triangleright_\tau \emptyset} \qquad \frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T} \qquad \frac{\Gamma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon \quad \Gamma \vdash \tau' : (\Upsilon_0;\Upsilon) \triangleright_\tau \Upsilon'}{\Gamma \vdash \tau;\tau' : \Upsilon_0 \triangleright_\tau \Upsilon;\Upsilon'}$$

Remark

type of a store $=$ *list of source types*

how *these types are translated* $=$ $\blacktriangleright$ $=$ ***parameter*** *of the target*

Stores

$\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

In this paper, we only use **lists** to represent stores:

Source types	A	$::=$	$X \mid A \to B$	$F ::= A \mid A^\perp$
Store types	Υ	$::=$	$Y \mid \emptyset \mid \Upsilon, F \mid \Upsilon;\Upsilon'$	
Stores	τ	$::=$	$\delta \mid [] \mid \tau[t] \mid \tau;\tau'$	

$\vdash \tau : \Upsilon' \triangleright_\tau \Upsilon$

"Appended to a store of type Υ', the store τ is of type Υ."

$$\frac{}{\Gamma \vdash [] : \emptyset \triangleright_\tau \emptyset} \qquad \frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T} \qquad \frac{\Gamma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon \quad \Gamma \vdash \tau' : (\Upsilon_0;\Upsilon) \triangleright_\tau \Upsilon'}{\Gamma \vdash \tau;\tau' : \Upsilon_0 \triangleright_\tau \Upsilon;\Upsilon'}$$

Remark

type of a store = *list of source types*

***how** these types are translated* = $\blacktriangleright$ = ***parameter** of the target*

Coercions

$\vdash \sigma : \Upsilon' <: \Upsilon$

Explicit witnesses of list inclusions:

1. Base case

$$\frac{}{\Gamma \vdash \varepsilon : \emptyset <: \emptyset}\,(\varepsilon)$$

2. Local identity

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \sigma^{+} : (\Upsilon', F) <: (\Upsilon, F)}\,(<:_{+})$$

3. Strict extension

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

Example:

$$\frac{\dots}{\vdash \Uparrow((\Uparrow \varepsilon)^{++}) : T_0, T, U, T_1 <: T, U}$$

Coercions

$\vdash \sigma : \Upsilon' <: \Upsilon$

Explicit witnesses of list inclusions:

1. Base case

$$\frac{}{\Gamma \vdash \varepsilon : \emptyset <: \emptyset}\,(\varepsilon)$$

2. Local identity

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \sigma^{+} : (\Upsilon', F) <: (\Upsilon, F)}\,(<:_{+})$$

3. Strict extension

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

Example:

$$\frac{\ldots}{\vdash \Uparrow ((\Uparrow \varepsilon)^{++}) : T_0, T, U, T_1 <: T, U}$$

Coercions

$\vdash \sigma : \Upsilon' <: \Upsilon$

Explicit witnesses of list inclusions:

1. Base case

$$\frac{}{\Gamma \vdash \varepsilon : \emptyset <: \emptyset}\,(\varepsilon)$$

2. Local identity

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \sigma^{+} : (\Upsilon', F) <: (\Upsilon, F)}\,(<:_{+})$$

3. Strict extension

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

Example:

$$\frac{\cdots}{\vdash \Uparrow((\Uparrow \varepsilon)^{++}) : T_0, T, U, T_1 <: T, U}$$

Coercions

$\vdash \sigma : \Upsilon' <: \Upsilon$

Explicit witnesses of list inclusions:

1. Base case

$$\frac{}{\Gamma \vdash \varepsilon : \emptyset <: \emptyset}\,(\varepsilon)$$

2. Local identity

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \sigma^{+} : (\Upsilon', F) <: (\Upsilon, F)}\,(<:_{+})$$

3. Strict extension

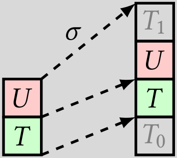
$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

Example:

$$\frac{\dots}{\vdash \Uparrow ((\Uparrow \varepsilon)^{++}) : T_0, T, U, T_1 <: T, U}$$

Remark: this corresponds to the function

- $0 \mapsto 1$
- $1 \mapsto 2$
- $2 \mapsto 4$



Coercions

$\vdash \sigma : \Upsilon' <: \Upsilon$

Explicit witnesses of list inclusions:

1. Base case

$$\frac{}{\Gamma \vdash \varepsilon : \emptyset <: \emptyset}\,(\varepsilon)$$

2. Local identity

$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \sigma^{+} : (\Upsilon', F) <: (\Upsilon, F)}\,(<:_{+})$$

3. Strict extension

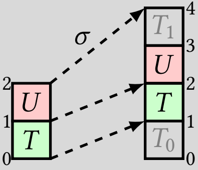
$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

Example:

$$\frac{\dots}{\vdash \Uparrow ((\Uparrow \varepsilon)^{++}) : T_0, T, U, T_1 <: T, U}$$

Remark: this corresponds to the function

- $0 \mapsto 1$
- $1 \mapsto 2$
- $2 \mapsto 4$



System F_Υ

In broad lines

System F extended with stores and coercions[1]

[1] Actually, false advertizing, the situation is more involved.

System F_Υ

Syntax: *Store type* Υ + *Stores* τ + *Coercions* σ +

$$\begin{array}{ll} \textit{Types} & T ::= X \mid T \to U \mid \Upsilon' <: \Upsilon \to T \mid \Upsilon \triangleright_\tau \Upsilon' \to T \mid \forall Y.T \\ \textit{Terms} & t ::= \boldsymbol{k} \mid x \mid \lambda x.t \mid t\,u \mid \lambda s.t \mid t\,\sigma \mid \lambda\delta.t \mid t\,\tau \mid \lambda Y.t \mid t\,\Upsilon \end{array}$$

$$\mid \mathtt{split}\ \tau\ \mathtt{at}\ n\ \mathtt{along}\ \sigma : \Upsilon' <: \Upsilon\ \mathtt{as}\ (Y_0, s_0, \delta_0), x, (Y_1, s_1, \delta_1)\ \mathtt{in}\ t$$

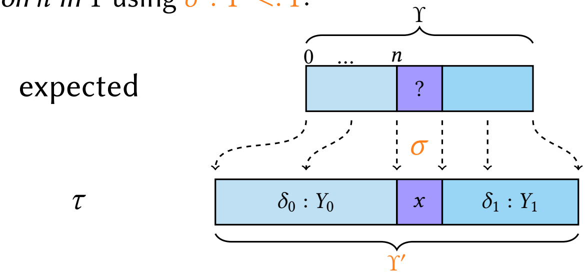
System F_Υ

Syntax: *Store type* Υ + *Stores* τ + *Coercions* σ +

$$\begin{array}{ll} \textit{Types} & T ::= X \mid T \to U \mid \Upsilon' <: \Upsilon \to T \mid \Upsilon \triangleright_\tau \Upsilon' \to T \mid \forall Y.T \\ \textit{Terms} & t ::= \boldsymbol{k} \mid x \mid \lambda x.t \mid t\ u \mid \lambda s.t \mid t\ \sigma \mid \lambda \delta.t \mid t\ \tau \mid \lambda Y.t \mid t\ \Upsilon \\ \end{array}$$

$\mid \texttt{split}\ \tau\ \texttt{at}\ n\ \texttt{along}\ \sigma : \Upsilon' <: \Upsilon\ \texttt{as}\ (Y_0, s_0, \delta_0), x, (Y_1, s_1, \delta_1)\ \texttt{in}\ t$

Intuitively, `split` allows to look in Υ' for the term *expected at position n in* Υ using $\sigma : \Upsilon' <: \Upsilon$:

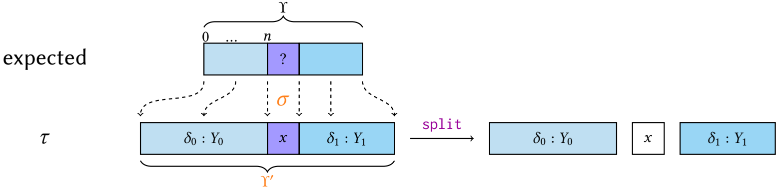


System F_Υ

Syntax: *Store type* Υ + *Stores* τ + *Coercions* σ +

$$
\begin{array}{ll}
\textit{Types} & T ::= X \mid T \to U \mid \Upsilon' <: \Upsilon \to T \mid \Upsilon \triangleright_\tau \Upsilon' \to T \mid \forall Y.T \\
\textit{Terms} & t ::= \boldsymbol{k} \mid x \mid \lambda x.t \mid t\,u \mid \lambda s.t \mid t\,\sigma \mid \lambda\delta.t \mid t\,\tau \mid \lambda Y.t \mid t\,\Upsilon \\
\mid\ \texttt{split}\ \tau\ \texttt{at}\ n\ \texttt{along}\ \sigma : \Upsilon' <: \Upsilon\ \texttt{as}\,(Y_0, s_0, \delta_0), x, (Y_1, s_1, \delta_1)\ \texttt{in}\ t &
\end{array}
$$

Intuitively, `split` allows to look in Υ' for the term *expected at position* n *in* Υ using $\sigma : \Upsilon' <: \Upsilon$:



System F_Υ

Syntax: *Store type* Υ + *Stores* τ + *Coercions* σ +

$$\begin{array}{ll} \textit{Types} & T ::= X \mid T \to U \mid \Upsilon' <: \Upsilon \to T \mid \Upsilon \triangleright_\tau \Upsilon' \to T \mid \forall Y.T \\ \textit{Terms} & t ::= \boldsymbol{k} \mid x \mid \lambda x.t \mid t\, u \mid \lambda s.t \mid t\, \sigma \mid \lambda \delta.t \mid t\, \tau \mid \lambda Y.t \mid t\, \Upsilon \\ \mid \texttt{split}\ \tau\ \texttt{at}\ n\ \texttt{along}\ \sigma : \Upsilon' <: \Upsilon\ \texttt{as}\,(Y_0, s_0, \delta_0), x, (Y_1, s_1, \delta_1)\ \texttt{in}\ t \end{array}$$

Three kinds of reductions:

- `split`
- normalization of coercions
- usual β-reduction

We have:

Properties

1. Reduction preserves typing *(Subject reduction)*
2. Typed terms normalize *(Normalization)*

Shallow embedding in Coq: `https://gitlab.com/emiquey/fupsilon`

Examples

In the paper, we take advantage of the genericity of F_Υ:

$$\frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T} \longleftarrow \boxed{\blacktriangleright \textit{ parameter depending on the translation}}$$

to define well-typed CEPS for simply-typed calculi:

✓ call-by-need ✓ call-by-name ✓ call-by-value

These translations exactly follow the intuitions we saw before:

negative translation *Kripke-style forcing*

Remark: we could also consider System F as source calculus, by changing the notion of source types.

Examples

In the paper, we take advantage of the genericity of F_Υ:

$$\frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T} \longleftarrow \boxed{\blacktriangleright \textit{ parameter depending on the translation}}$$

to define well-typed CEPS for simply-typed calculi:

✓ call-by-need ✓ call-by-name ✓ call-by-value

These translations exactly follow the intuitions we saw before:

negative translation *Kripke-style forcing*

Remark: we could also consider System F as source calculus, by changing the notion of source types.

Examples

In the paper, we take advantage of the genericity of F_Υ:

$$\frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T}$$

▶ *parameter depending on the translation*

to define well-typed CEPS for simply-typed calculi:

✓ call-by-need ✓ call-by-name ✓ call-by-value

These translations exactly follow the intuitions we saw before:

continuation-passing	+	environment-passing
negative translation		*Kripke-style forcing*

Remark: we could also consider System F as source calculus, by changing the notion of source types.

Examples

In the paper, we take advantage of the genericity of F_Υ:

$$\frac{\Gamma \vdash t : \Upsilon_0 \blacktriangleright T}{\Gamma \vdash [t] : \Upsilon_0 \triangleright_\tau T} \longleftarrow$$

▶ *parameter depending on the translation*

to define well-typed CEPS for simply-typed calculi:

✓ call-by-need ✓ call-by-name ✓ call-by-value

These translations exactly follow the intuitions we saw before:

continuation-passing	+	environment-passing
negative translation		*Kripke-style forcing*

Remark: we could also consider System F as source calculus, by changing the notion of source types.

Conclusion

We isolated the **key ingredients** for well-typed CEPS:

1. terms to represent and manipulate **typed stores**,
2. explicit **coercions** to witness store extensions.

F_Υ has the benefits of being **parametric**:

- suitable for CEPS with different evaluation strategies
- compatible with different sources/type systems.
- compatible with different implementation of stores

Conclusion

We isolated the **key ingredients** for well-typed CEPS:

1. terms to represent and manipulate **typed stores**,
2. explicit **coercions** to witness store extensions.

F_Υ has the benefits of being **parametric**:

- suitable for CEPS with different evaluation strategies
- compatible with different sources/type systems.
- compatible with different implementation of stores

Conclusion

From a logical viewpoint:

CEPS $\cong$ Kripke forcing interleaved with a negative translation

Connection between **forcing and environment** already known:

$$\begin{tikzcd} & \text{Presheaves} \arrow[dl, bend right, leftrightarrow] \arrow[dr, bend left, leftrightarrow] & \\ \text{Forcing} \arrow[rr, bend right, leftrightarrow] & & \text{State monad} \end{tikzcd}$$

Conclusion

Open questions / further work

1. Towards well-typed compilation transformations for lazily-evaluated calculi? (cf. MetaCoq project)
2. Exact expressiveness of F_Υ?
3. Type translation as a modality?

Conclusion

Open questions / further work

1. Towards well-typed compilation transformations for lazily-evaluated calculi? (cf. MetaCoq project)
2. Exact expressiveness of F_Υ?
3. Type translation as a modality?

Conclusion

Open questions / further work

1. Towards well-typed compilation transformations for lazily-evaluated calculi? (cf. MetaCoq project)
2. Exact expressiveness of F_Υ?
3. Type translation as a modality?

$$\boxed{\cdot \rhd_{\mathrm{t}} A \text{ is a function : store type} \mapsto \text{type}}$$

Conclusion

Open questions / further work

1. Towards well-typed compilation transformations for lazily-evaluated calculi? (cf. MetaCoq project)

2. Exact expressiveness of F_Υ?

3. Type translation as a modality?

$\boxed{\cdot \triangleright_{\mathsf{t}} A \text{ is a function : store type} \mapsto \text{type}}$

$$\begin{array}{rcl} \Box\mathcal{F} & \triangleq & \Upsilon \;\mapsto\; \forall \Upsilon' <: \Upsilon . \Upsilon' \to (\mathcal{F}\,\Upsilon') \to \bot \\ \cdot \triangleright_{\mathsf{t}} A & = & \Box(\cdot \triangleright_{\mathsf{E}} A) = \Box(\Box(\cdot \triangleright_{\mathsf{V}} A)) = \ldots \end{array}$$

Thank you for your attention.