CPS translations & environments

A well-typed story

Hugo Herbelin
Étienne Miquey

Inria
Université de Paris, IRIF

LIP
ÉNS de Lyon

March 19th 2021

ENS DE LYON

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**

↬ *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**

↬ *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**

↬ *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*

In praise of laziness

Call-by-need evaluation strategy:

- evaluates arguments of functions only when needed

  $\looparrowright$ *as in call-by-name*

- shares the evaluations across all places where they are needed

  $\looparrowright$ *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
  $\looparrowright$ *as in call-by-name*
- shares the evaluations across all places where they are needed
  $\looparrowright$ *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)

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

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

Milner Abstract Machine (CbName)

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

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)

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

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

Milner Abstract Machine (CbName)

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

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)

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

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

Milner Abstract Machine (CbName)

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

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?

Classical logic and control operators

Classical logic:

Intuitionistic logic $\quad + \quad A \vee \neg A$

(or $\neg\neg A \to A$, $((A \to B) \to A) \to A$, etc.

Classical Curry-Howard:

λ-calculus $\quad + \quad$ `call/cc`

(Griffin'90: `call/cc` $: \forall AB.((A \to B) \to A) \to A$*)*

Continuation-passing style translation:

- operational semantics for `call/cc`
- Gödel's negative translation

Classical call-by-need

```
let a = call/cc (λk.(I,λx.throw k x))
    f = fst a
    q = snd a
in f q (I,I)
```

How should a call-by-need strategy compute?

Classical call-by-need

```
let a = call/cc (λk.(I,λx.throw k x))
    f = fst a
    q = snd a
in f q (I,I)
```

How should a call-by-need strategy compute?

- Okasaki, Lee, Tarditi'94:

  Only the chain of bindings forcing an effect are not shared.

```
let a = (I,λx.throw k x)
    f = I
    q = λx.throw k x
in q (I,I)
```

$\longrightarrow$

```
let a = (I,I)
    f = fst a
    q = λx.throw k x
in f q (I,I)
```

$\rightarrow$ *loops forever...*

Classical call-by-need

```
let a = call/cc (λk.(I,λx.throw k x))
    f = fst a
    q = snd a
in f q (I,I)
```

How should a call-by-need strategy compute?

- Ariola *et al.*'12:

None of the bindings inside a side-effect are shared.

```
let a = (I,λx.throw k x)
    f = I
    q = λx.throw k x
in throw k (I,I)
```

$\longrightarrow$

```
let a = (I,I)
    f = fst a
    q = snd a
in f q (I,I)
```

$\rightarrow$ (I,I)

This talk

Ariola *et al.*'12:

- defined a call-by-need sequent calculus $\overline{\lambda}_{[lv\tau\star]}$
- used *Danvy's semantics artifacts* to derive an untyped CPS

Goal #1

Do simply-typed terms of $\overline{\lambda}_{[lv\tau\star]}$ normalize?

Goal #2

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

This talk

Goal #2

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 talk

Goal #2

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.

- We use it to implement simply-typed CEPS translations for:

This talk

Goal #2

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

Our toolbox

Danvy's semantics artifacts & Krivine realizability

CPS translation

Continuation-passing style translation: $[\![\cdot]\!] : \mathit{source} \longrightarrow \lambda^{\text{something}}$

- preserving reduction

$$t \xrightarrow{1} t' \qquad \Rightarrow \qquad [\![t]\!] \xrightarrow{+} [\![t']\!]$$

- preserving typing

$$\Gamma \vdash t : A \qquad \Rightarrow \qquad [\![\Gamma]\!] \vdash [\![t]\!] : [\![A]\!]$$

- the type $[\![\bot]\!]$ is not inhabited

Benefits

If $\lambda^{\text{something}}$ is sound and normalizing:

1. If $[\![t]\!]$ normalizes, then t normalizes
2. If t is typed, then t normalizes
3. The source language is sound, *i.e.* there is no term $\vdash t : \bot$

An atomic vision of logic

P.A. Melliès (2009) :

> logic... leading to the decomposition of logical connectives and modalities into smaller meaningful components. This practice has been extremely fruitful in the past, and leads to the bold idea that there are such things as
>
> **elementary particles of logic**
>
> whose combined properties and interactions produce the logical phenomenon.

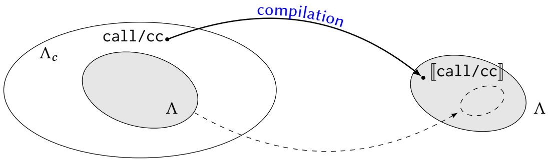
Atomism, computationally:

An atomic vision of logic

P.A. Melliès (2009) :

> logic... leading to the decomposition of logical connectives and modalities into smaller meaningful components. This practice has been extremely fruitful in the past, and leads to the bold idea that there are such things as
>
> **elementary particles of logic**
>
> whose combined properties and interactions produce the logical phenomenon.

Atomism, computationally:

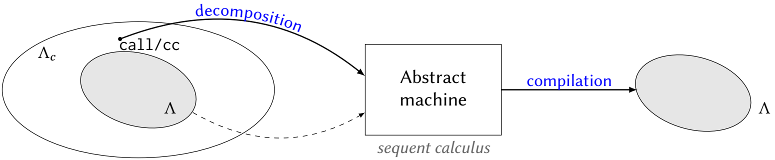


An atomic vision of logic

P.A. Melliès (2009) :

> logic... leading to the decomposition of logical connectives and modalities into smaller meaningful components. This practice has been extremely fruitful in the past, and leads to the bold idea that there are such things as
>
> **elementary particles of logic**
>
> whose combined properties and interactions produce the logical phenomenon.

Atomism, computationally:



Danvy's semantic artifacts *[Tool 1]*

Defunctionalized Interpreters for Call-by-Need Evaluation
Danvy et al. (2010)

A methodology for reductionism

1. an operational semantics
2. a small-step calculus or abstract machine
3. a continuation-passing style translation
4. a realizability model

Coming next: this method on an easy example

Danvy's semantic artifacts *[Tool 1]*

Defunctionalized Interpreters for Call-by-Need Evaluation
Danvy et al. (2010)

A methodology for reductionism

1. an operational semantics
2. a small-step calculus or abstract machine
3. a continuation-passing style translation
4. a realizability model

Coming next: this method on an easy example

The $\lambda\mu\tilde{\mu}$-calculus

The duality of computation
Curien/Herbelin (2000)

Syntax:

$$\begin{array}{ll} \textit{Terms} & t ::= x \mid \lambda x.t \mid \mu\alpha.c \\ \textit{Contexts} & e ::= \alpha \mid t\cdot e \mid \tilde{\mu}x.c \\ \textit{Commands} & c ::= \langle t \,\|\, e\rangle \end{array}$$

Typing rules:

$$\frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta}{\langle t \,\|\, e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A \mid \Delta} \qquad \frac{\Gamma, x : A \vdash t : B \mid \Delta}{\Gamma \vdash \lambda x.t : A \to B \mid \Delta} \qquad \frac{c : (\Gamma \vdash \Delta, \alpha : A)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}$$

$$\frac{(\alpha : A) \in \Delta}{\Gamma \mid \alpha : A \vdash \Delta} \qquad \frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid t\cdot e : A \to B \vdash \Delta} \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Delta}$$

The $\lambda\mu\tilde{\mu}$-calculus

The duality of computation
Curien/Herbelin (2000)

Syntax:

$$
\begin{array}{ll}
\textit{Terms} & t ::= x \mid \lambda x.t \mid \mu\alpha.c \\
\textit{Contexts} & e ::= \alpha \mid t \cdot e \mid \tilde{\mu}x.c \\
\textit{Commands} & c ::= \langle t \,\|\, e \rangle
\end{array}
$$

Typing rules:

$$\frac{\Gamma \vdash \quad A \mid \Delta \qquad \Gamma \mid \quad A \vdash \Delta}{(\Gamma \vdash \Delta)}$$

$$\frac{A \quad \in \Gamma}{\Gamma \vdash \quad A \mid \Delta} \qquad \frac{\Gamma, \quad A \vdash \quad B \mid \Delta}{\Gamma \vdash \quad A \to B \mid \Delta} \qquad \frac{\Gamma \vdash \Delta, \quad A}{\Gamma \vdash \quad A \mid \Delta}$$

$$\frac{A \quad \in \Delta}{\Gamma \mid \quad A \vdash \Delta} \qquad \frac{\Gamma \vdash \quad A \mid \Delta \qquad \Gamma \mid \quad B \vdash \Delta}{\Gamma \mid \quad A \to B \vdash \Delta} \qquad \frac{\Gamma, \quad A \vdash \Delta}{\Gamma \mid \quad A \vdash \Delta}$$

Call-by-value $\lambda\mu\tilde{\mu}$-calculus

Syntax:

Terms	$t ::= V \mid \mu\alpha.c$	*Contexts*	$e ::= E \mid \tilde{\mu}x.c$
Values	$V ::= x \mid \lambda x.t$	*Co-values*	$E ::= \alpha \mid t \cdot e$

Commands $\quad c ::= \langle t \| e \rangle$

Reduction rules:

$$\begin{array}{ccc}
\langle \mu\alpha.c \| e \rangle & \to & c[e/\alpha] \\
\langle V \| \tilde{\mu}x.c \rangle & \to & c[V/x] \\
\langle \lambda x.t \| u \cdot e \rangle & \to & \langle u \| \tilde{\mu}x.\langle t \| e \rangle \rangle
\end{array}$$

Semantic artifacts

$$\begin{array}{lll lll}
\textit{Terms} & t & ::= V \mid \mu\alpha.c & \textit{Contexts} & e & ::= E \mid \tilde{\mu}x.c \\
\textit{Values} & V & ::= x \mid \lambda x.t & \textit{Co-values} & E & ::= \alpha \mid t \cdot e
\end{array}$$

$$\textit{Commands} \quad c ::= \langle t \,\|\, e \rangle$$

Small steps

$$\begin{array}{lrcl}
\mathrm{t} & \langle \mu\alpha.c \,\|\, e\rangle_{\mathrm{t}} & \rightsquigarrow & c_{\mathrm{t}}[e/\alpha] \\
 & \langle V \,\|\, e\rangle_{\mathrm{t}} & \rightsquigarrow & \langle V \,\|\, e\rangle_{\mathrm{e}} \\
\mathrm{e} & \langle V \,\|\, \tilde{\mu}x.c\rangle_{\mathrm{e}} & \rightsquigarrow & c_{\mathrm{e}}[V/x] \\
 & \langle V \,\|\, u \cdot e\rangle_{\mathrm{e}} & \rightsquigarrow & \langle V \,\|\, u \cdot e\rangle_{\mathrm{V}} \\
\mathrm{V} & \langle \lambda x.t \,\|\, u \cdot e\rangle_{\mathrm{V}} & \rightsquigarrow & \langle u \,\|\, \tilde{\mu}x.\langle t \,\|\, e\rangle\rangle_{\mathrm{t}}
\end{array}$$

Semantic artifacts

$$\begin{array}{llll} \textit{Terms} & t ::= V \mid \mu\alpha.c & \textit{Contexts} & e ::= E \mid \tilde{\mu}x.c \\ \textit{Values} & V ::= x \mid \lambda x.t & \textit{Co-values} & E ::= \alpha \mid t \cdot e \end{array}$$

$$\textit{Commands} \quad c ::= \langle t \parallel e \rangle$$

	Small steps			**CPS**
t	$\langle \mu\alpha.c \parallel e \rangle_{\mathrm{t}}$	$\rightsquigarrow$	$c_{\mathrm{t}}[e/\alpha]$	$[\![\mu\alpha.c]\!]_{\mathrm{t}} \triangleq \lambda e.(\lambda\alpha.[\![c]\!]_{\mathrm{c}})\, e$
	$\langle V \parallel e \rangle_{\mathrm{t}}$	$\rightsquigarrow$	$\langle V \parallel e \rangle_{\mathrm{e}}$	$[\![V]\!]_{\mathrm{t}} \triangleq \lambda e.e\, [\![V]\!]_{\mathrm{V}}$
e	$\langle V \parallel \tilde{\mu}x.c \rangle_{\mathrm{e}}$	$\rightsquigarrow$	$c_{\mathrm{e}}[V/x]$	$[\![\tilde{\mu}x.c]\!]_{\mathrm{e}} \triangleq \lambda V.(\lambda x.[\![c]\!]_{\mathrm{c}})\, V$
	$\langle V \parallel u \cdot e \rangle_{\mathrm{e}}$	$\rightsquigarrow$	$\langle V \parallel u \cdot e \rangle_{\mathrm{V}}$	$[\![u \cdot e]\!]_{\mathrm{e}} \triangleq \lambda V.V\, [\![u]\!]_{t}\, [\![e]\!]_{\mathrm{e}}$
V	$\langle \lambda x.t \parallel u \cdot e \rangle_{\mathrm{V}}$	$\rightsquigarrow$	$\langle u \parallel \tilde{\mu}x.\langle t \parallel e \rangle \rangle_{\mathrm{t}}$	$[\![\lambda x.t]\!]_{\mathrm{V}} \triangleq \lambda ue.u\, (\lambda x.[\![t]\!]_{\mathrm{t}}\, e)$

$$c \overset{1}{\rightsquigarrow} c' \quad \Rightarrow \quad [\![c]\!]_{\mathrm{c}} \xrightarrow{+}_{\beta} [\![c']\!]_{\mathrm{c}}$$

Semantic artifacts

$$\begin{array}{llll} \textit{Terms} & t ::= V \mid \mu\alpha.c & \textit{Contexts} & e ::= E \mid \tilde{\mu}x.c \\ \textit{Values} & V ::= x \mid \lambda x.t & \textit{Co-values} & E ::= \alpha \mid t \cdot e \end{array}$$

$$\textit{Commands} \quad c ::= \langle t \| e \rangle$$

CPS

t

$$[\![\mu\alpha.c]\!]_{\mathsf{t}} \triangleq \lambda e.(\lambda\alpha.[\![c]\!]_{\mathsf{c}})\, e$$
$$[\![V]\!]_{\mathsf{t}} \triangleq \lambda e.e\, [\![V]\!]_{\mathsf{V}}$$

e

$$[\![\tilde{\mu}x.c]\!]_{\mathsf{e}} \triangleq \lambda V.(\lambda x.[\![c]\!]_{\mathsf{c}})\, V$$
$$[\![u \cdot e]\!]_{\mathsf{e}} \triangleq \lambda V.V\, [\![u]\!]_{t}\, [\![e]\!]_{\mathsf{e}}$$

V

$$[\![\lambda x.t]\!]_{\mathsf{V}} \triangleq \lambda ue.u\, (\lambda x.[\![t]\!]_{\mathsf{t}}\, e)$$

Types translation

$$[\![A]\!]_{\mathsf{t}} \triangleq [\![A]\!]_{\mathsf{e}} \to \bot$$
$$[\![A]\!]_{\mathsf{e}} \triangleq [\![A]\!]_{\mathsf{V}} \to \bot$$
$$[\![A \to B]\!]_{\mathsf{V}} \triangleq [\![A]\!]_{\mathsf{t}} \to [\![A]\!]_{\mathsf{e}} \to \bot$$

$$\Gamma \vdash t : A \mid \Delta \qquad \Rightarrow \qquad [\![\Gamma]\!]_{\mathsf{V}}, [\![\Delta]\!]_{\mathsf{e}} \vdash [\![t]\!]_{\mathsf{t}} : [\![A]\!]_{\mathsf{t}}$$

Realizability *à la* Krivine *[Tool 2]*

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $\perp\!\!\!\perp$: commands, referee

$$\langle t \,\|\, e\rangle > c_0 > \cdots > c_n \in \perp\!\!\!\perp ?$$

$\rightsquigarrow \perp\!\!\!\perp \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{\perp\!\!\!\perp} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \,\|\, e\rangle \in \perp\!\!\!\perp\}$$

Realizability *à la* Krivine *[Tool 2]*

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $⫫$: commands, referee

$$\langle t \| e \rangle > c_0 > \cdots > c_n \in ⫫ ?$$

$\rightsquigarrow$ $⫫ \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{⫫} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \| e \rangle \in ⫫\}$$

Realizability *à la* Krivine *[Tool 2]*

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $\perp\!\!\!\perp$: commands, referee

$$\langle t \,\|\, e\rangle \succ c_0 \succ \cdots \succ c_n \in \perp\!\!\!\perp ?$$

$\rightsquigarrow \perp\!\!\!\perp \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{\perp\!\!\!\perp} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \,\|\, e\rangle \in \perp\!\!\!\perp\}$$

Realizability *à la* Krivine *[Tool 2]*

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $⫫$: commands, referee

$$\langle t \parallel e \rangle \succ c_0 \succ \cdots \succ c_n \in ⫫?$$

$\rightsquigarrow ⫫ \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{⫫} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \parallel e \rangle \in ⫫\}$$

Realizability *à la* Krivine *[Tool 2]*

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $\perp\!\!\!\perp$: commands, referee

$$\langle t \parallel e \rangle \succ c_0 \succ \cdots \succ c_n \in \perp\!\!\!\perp ?$$

$\rightsquigarrow$ $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{\perp\!\!\!\perp} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \parallel e \rangle \in \perp\!\!\!\perp\}$$

Realizability *à la* Krivine [Tool 2]

Intuition

- falsity value $\|A\|$: contexts, opponent to A
- truth value $|A|$: terms, player of A
- pole $⫫$: commands, referee

$$\langle t \| e \rangle \succ c_0 \succ \cdots \succ c_n \in ⫫ ?$$

$\rightsquigarrow$ $⫫ \subseteq \Lambda \star \Pi$ closed by anti-reduction

Truth value defined by **orthogonality** :

$$|A| = \|A\|^{⫫} = \{t \in \Lambda : \forall e \in \|A\|, \langle t \| e \rangle \in ⫫\}$$

Semantic artifacts++

$$\begin{array}{llll}
\textit{(Terms)} & t ::= \mu\alpha.c \mid x \mid V & \textit{(Contexts)} & e ::= \tilde\mu x.c \mid E \\
\textit{(Values)} & V ::= \lambda x.t & \textit{(Co-values)} & E ::= \alpha \mid u \cdot e
\end{array}$$

Small steps | **Realizability**

$$\begin{array}{clclc}
\mathrm{t} & \langle \mu\alpha.c \parallel e\rangle_{\mathrm{t}} & \rightsquigarrow & c_{\mathrm{t}}[e/\alpha] & |A|_{\mathrm{t}} \triangleq \|A\|_{\mathrm{E}}{}^{\perp\!\!\perp} \\
 & \langle V \parallel e\rangle_{\mathrm{t}} & \rightsquigarrow & \langle V \parallel e\rangle_{\mathrm{e}} & \\
\mathrm{e} & \langle V \parallel \tilde\mu x.c\rangle_{\mathrm{e}} & \rightsquigarrow & c_{\mathrm{e}}[V/x] & \|A\|_{\mathrm{e}} \triangleq |A|_{\mathrm{V}}{}^{\perp\!\!\perp} \\
 & \langle V \parallel u \cdot e\rangle_{\mathrm{e}} & \rightsquigarrow & \langle V \parallel u \cdot e\rangle_{\mathrm{V}} & \\
\mathrm{V} & \langle \lambda x.t \parallel u \cdot e\rangle_{\mathrm{V}} & \rightsquigarrow & \langle u \parallel \tilde\mu x.\langle t \parallel e\rangle\rangle_{\mathrm{t}} & |A \to B|_{\mathrm{V}} \triangleq \{\lambda x.t : \forall V \in |A|_{\mathrm{V}}, t[V/x] \in |B|_{\mathrm{t}}\}
\end{array}$$

Adequacy

For any pole $\perp\!\!\perp$, if $\sigma \Vdash \Gamma \cup \Delta$, then:

1. $\Gamma \vdash t : A \mid \Delta \;\Rightarrow\; t[\sigma] \in |A|_{\mathrm{t}}$
2. $\Gamma \mid e : A \vdash \Delta \;\Rightarrow\; e[\sigma] \in \|A\|_{\mathrm{e}}$
3. $c : (\Gamma \vdash \Delta) \;\Rightarrow\; c[\sigma] \in \perp\!\!\perp$

Semantic artifacts++

(Terms)	$t ::= \mu\alpha.c \mid x \mid V$	*(Contexts)*	$e ::= \tilde{\mu}x.c \mid E$
(Values)	$V ::= \lambda x.t$	*(Co-values)*	$E ::= \alpha \mid u \cdot e$

Small steps

t
$$\langle \mu\alpha.c \,\|\, e\rangle_t \rightsquigarrow c_t[e/\alpha]$$
$$\langle V \,\|\, e\rangle_t \rightsquigarrow \langle V \,\|\, e\rangle_e$$

e
$$\langle V \,\|\, \tilde{\mu}x.c\rangle_e \rightsquigarrow c_e[V/x]$$
$$\langle V \,\|\, u\cdot e\rangle_e \rightsquigarrow \langle V \,\|\, u\cdot e\rangle_V$$

V
$$\langle \lambda x.t \,\|\, u\cdot e\rangle_V \rightsquigarrow \langle u \,\|\, \tilde{\mu}x.\langle t \,\|\, e\rangle\rangle_t$$

Realizability

$$|A|_t \triangleq \|A\|_E^{\perp\!\!\!\perp}$$

$$\|A\|_e \triangleq |A|_V^{\perp\!\!\!\perp}$$

$$|A \to B|_V \triangleq \{\lambda x.t : \forall V \in |A|_V, t[V/x] \in |B|_t\}$$

Adequacy

For any pole $\perp\!\!\!\perp$, if $\sigma \Vdash \Gamma \cup \Delta$, then:

1. $\Gamma \vdash t : A \mid \Delta \quad\Rightarrow\quad t[\sigma] \in |A|_t$
2. $\Gamma \mid e : A \vdash \Delta \quad\Rightarrow\quad e[\sigma] \in \|A\|_e$
3. $c : (\Gamma \vdash \Delta) \quad\Rightarrow\quad c[\sigma] \in \perp\!\!\!\perp$

Results

Normalizing commands

$\perp\!\!\!\perp_{\Downarrow} \triangleq \{c : c \text{ normalizes}\}$ defines a valid pole.

Proof. If $c \to c'$ and c' normalizes, so does c. □

Normalization

For any command c, if $c : \Gamma \vdash \Delta$, then c normalizes.

Proof. By adequacy, any typed command c belongs to the pole $\perp\!\!\!\perp_{\Downarrow}$. □

Soundness

There is no term t such that $\vdash t : \perp \mid$.

Proof. Otherwise, $t \in |\perp|_t = \Pi^{\perp\!\!\!\perp}$ for any pole, absurd ($\perp\!\!\!\perp \triangleq \emptyset$). □

Normalization of classical call-by-need

Realizability interpretation of $\overline{\lambda}_{[lv\tau\star]}$

The $\overline{\lambda}_{[lv\tau\star]}$-calculus

(Analyzing Ariola *et al.* '12)

Sequent calculus:

$$\langle t \,\|\, e \rangle \tau$$

$$\begin{array}{ccc} \nearrow & \uparrow & \nwarrow \\ \text{Term} & \text{Context} & \text{Environment} \end{array}$$

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.	$E ::= F \mid \alpha \mid \tilde{\mu}[x].\langle x \,\|\, F\rangle\tau$
Forcing cont.	$F ::= t \cdot E \mid \boldsymbol{\kappa}$

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

The $\overline{\lambda}_{[lv\tau\star]}$-calculus

(Analyzing Ariola *et al.* '12)

Sequent calculus:

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

Term ↗ Context ↑ Environment ↖

Syntax

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

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	$c ::= \langle t \parallel e \rangle$

The $\overline{\lambda}_{[lv\tau\star]}$-calculus

(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.	$E ::= F \mid \alpha \mid \tilde{\mu}[x].\langle x \| F\rangle\tau$
Forcing cont.	$F ::= t \cdot E \mid \boldsymbol{\kappa}$

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

Lazy reduction:

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

Typing stores

Stores are typed with typing hypotheses Γ

$$\frac{\Gamma,\Gamma' \vdash_c c \qquad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l c\tau}\ (l)$$

$$\frac{\Gamma \vdash_\tau \tau : \Gamma' \quad \Gamma,\Gamma' \vdash_t t : A}{\Gamma \vdash_\tau \tau[x := t] : \Gamma', x : A}\ (\tau_t)$$

Semantic artifacts

Classical Call-by-Need: ...
Ariola et al. [2012]

Small steps:

$$
\begin{array}{llcl}
e & \langle t \,\|\, \tilde\mu x.c\rangle_e \tau & \to & c_e \tau[x := t] \\
 & \langle t \,\|\, E\rangle_e \tau & \to & \langle t \,\|\, E\rangle_t \tau \\
t & \langle \mu\alpha.c \,\|\, E\rangle_t \tau & \to & (c_e[E/\alpha])\tau \\
 & \langle V \,\|\, E\rangle_t \tau & \to & \langle V \,\|\, E\rangle_E \tau \\
E & \langle V \,\|\, \tilde\mu[x].\langle x \,\|\, F\rangle\tau'\rangle_E \tau & \to & \langle V \,\|\, F\rangle_V \tau[x := V]\tau' \\
 & \langle V \,\|\, F\rangle_E \tau & \to & \langle V \,\|\, F\rangle_V \tau \\
V & \langle x \,\|\, F\rangle_V \tau[x := t]\tau' & \to & \langle t \,\|\, \tilde\mu[x].\langle x \,\|\, F\rangle\tau'\rangle_t \tau \\
 & \langle \lambda x.t \,\|\, F\rangle_V \tau & \to & \langle \lambda x.t \,\|\, F\rangle_F \tau \\
F & \langle \lambda x.t \,\|\, u \cdot E\rangle_F \tau & \to & \langle u \,\|\, \tilde\mu x.\langle t \,\|\, E\rangle\rangle_e \tau
\end{array}
$$

Semantic artifacts

Classical Call-by-Need: ...
Ariola et al. [2012]

CPS :

$$\llbracket \langle t \parallel e \rangle \tau \rrbracket := \llbracket e \rrbracket_{\mathsf{e}}\ \llbracket \tau \rrbracket_{\tau}\ \llbracket t \rrbracket_{\mathsf{t}}$$

e

$$\llbracket \tilde{\mu} x.c \rrbracket_{\mathsf{e}} := \lambda \tau t.\llbracket c \rrbracket_{\mathsf{c}}\ \tau[x := t]$$

$$\llbracket E \rrbracket_{\mathsf{e}} := \lambda \tau t.t\ \tau\ \llbracket E \rrbracket_{\mathsf{E}}$$

t

$$\llbracket \mu \alpha.c \rrbracket_{\mathsf{t}} := \lambda \tau E.(\llbracket c \rrbracket_{\mathsf{c}}\ \tau)[E/\alpha]$$

$$\llbracket V \rrbracket_{\mathsf{t}} := \lambda \tau E.E\ \tau\ \llbracket V \rrbracket_{\mathsf{V}}$$

E

$$\llbracket \tilde{\mu}[x].\langle x \parallel F \rangle \tau' \rrbracket_{\mathsf{E}} := \lambda \tau V.V\ \tau[x := V]\tau'\ \llbracket F \rrbracket_{\mathsf{f}}$$

$$\llbracket F \rrbracket_{\mathsf{E}} := \lambda \tau V.V\ \tau\ \llbracket F \rrbracket_{\mathsf{f}}$$

V

$$\llbracket x \rrbracket_{\mathsf{V}} := \lambda \tau F.\tau(x)\ \tau\ (\lambda \tau V.V\ \tau[x := V]\tau'\ \llbracket F \rrbracket_{\mathsf{f}})$$

$$\llbracket \lambda x.t \rrbracket_{\mathsf{V}} := \lambda \tau F.F\ \tau\ (\lambda u \tau E.\llbracket t \rrbracket_{\mathsf{t}}\ \tau[x := u]\ E)$$

F

$$\llbracket u \cdot E \rrbracket_{\mathsf{f}} := \lambda \tau v.v\ \llbracket t \rrbracket_{\mathsf{t}}\ \tau\ \llbracket E \rrbracket_{\mathsf{E}}$$

Semantic artifacts

Realizability interpretation and normalization of typed ...
M. & Herbelin [2018]

Small-step:

$$
\begin{array}{lrll}
e & \langle t \parallel \tilde\mu x.c\rangle_e \tau & \to & \ldots \\
 & \langle t \parallel E\rangle_e \tau & \to & \ldots \\
t & \langle \mu\alpha.c \parallel E\rangle_t \tau & \to & \ldots \\
 & \langle V \parallel E\rangle_t \tau & \to & \ldots \\
E & \langle V \parallel \tilde\mu[x].\langle x \parallel F\rangle\tau'\rangle_E \tau & \to & \ldots \\
 & \langle V \parallel F\rangle_E \tau & \to & \ldots \\
V & \langle x \parallel F\rangle_V \tau[x := t]\tau' & \to & \ldots \\
 & \langle v \parallel F\rangle_V \tau & \to & \ldots \\
F & \langle v \parallel u \cdot E\rangle_F \tau & \to & \ldots \\
v & \langle \lambda x.t \parallel u \cdot E\rangle_v \tau & \to & \ldots
\end{array}
$$

Realizability:

$$(\perp\!\!\!\perp \subseteq \Lambda \times \Pi \times \tau)$$

$$
\begin{array}{rl}
\|A\|_e := & \{\ e? \in |A|_t{}^{\perp\!\!\perp}\} \\
|A|_t := & \{\ t? \in \|A\|_E{}^{\perp\!\!\perp}\} \\
\|A\|_E := & \{\ E? \in |A|_V{}^{\perp\!\!\perp}\} \\
|A|_V := & \{\ V? \in \|A\|_F{}^{\perp\!\!\perp}\} \\
\|A\|_F := & \{\ F? \in |A|_v{}^{\perp\!\!\perp}\} \\
|A \to B|_v := & \{\ \lambda x.t? \ :\ u? \in |A|_t \\
 & \quad \Rightarrow t[u/x]? \in |B|_t\}
\end{array}
$$

Semantic artifacts

Realizability interpretation and normalization of typed ...
M. & Herbelin [2018]

Small-step:

e	$\langle t \parallel \tilde\mu x.c\rangle_e \tau \to \dots$	
	$\langle t \parallel E\rangle_e \tau \to \dots$	
t	$\langle \mu\alpha.c \parallel E\rangle_t \tau \to \dots$	
	$\langle V \parallel E\rangle_t \tau \to \dots$	
E	$\langle V \parallel \tilde\mu[x].\langle x \parallel F\rangle\tau'\rangle_E \tau \to \dots$	
	$\langle V \parallel F\rangle_E \tau \to \dots$	
V	$\langle x \parallel F\rangle_V \tau[x := t]\tau' \to \dots$	
	$\langle v \parallel F\rangle_V \tau \to \dots$	
F	$\langle v \parallel u \cdot E\rangle_F \tau \to \dots$	
v	$\langle \lambda x.t \parallel u \cdot E\rangle_v \tau \to \dots$	

Realizability:

$$(\perp\!\!\!\perp \subseteq \Lambda \times \Pi \times \tau)$$

$$\|A\|_{\mathsf{e}} := \{(e|\tau) \in |A|_{\mathsf{t}}{}^{\perp\!\!\!\perp}\}$$

$$|A|_{\mathsf{t}} := \{(t|\tau) \in \|A\|_{\mathsf{E}}{}^{\perp\!\!\!\perp}\}$$

$$\|A\|_{\mathsf{E}} := \{(E|\tau) \in |A|_{\mathsf{V}}{}^{\perp\!\!\!\perp}\}$$

$$|A|_{\mathsf{V}} := \{(V|\tau) \in \|A\|_{\mathsf{F}}{}^{\perp\!\!\!\perp}\}$$

$$\|A\|_{\mathsf{F}} := \{(F|\tau) \in |A|_{\mathsf{v}}{}^{\perp\!\!\!\perp}\}$$

$$|A \to B|_{\mathsf{v}} := \{(\lambda x.t|\tau) : (u|\tau') \in |A|_{\mathsf{t}} \Rightarrow (t|\overline{\tau\tau'}[x := u]) \in |B|_{\mathsf{t}}\}$$

Realizability interpretation

Key ideas

- **Term-in-store** $(t|\tau)$: $FV(t) \subseteq \mathrm{dom}(\tau)$ (τ closed)
  generalizes closed terms
- **Pole** : set of closures $\perp\!\!\!\perp$ which is:
  - *closed by anti-reduction*:

$$c'\tau' \in \perp\!\!\!\perp \quad \text{and} \quad c\tau \to c'\tau' \quad \text{implies} \quad c\tau \in \perp\!\!\!\perp$$

  - *closed by store extension*:

$$c\tau \in \perp\!\!\!\perp \quad \text{and} \quad \tau \triangleleft \tau' \quad \text{implies} \quad c\tau' \in \perp\!\!\!\perp$$

- **Orthogonality** :

$$(t|\tau)\perp\!\!\!\perp(e|\tau') \triangleq \quad \tau, \tau' \text{ compatible} \quad \wedge \quad \langle t \,\|\, e\rangle\overline{\tau\tau'} \in \perp\!\!\!\perp.$$

- **Realizers**: definitions derived from the small-step rules!

Realizability interpretation

Adequacy

For all $\perp\!\!\!\perp$, if $\tau \Vdash \Gamma$ and $\Gamma \vdash_c c$, then $c\tau \in \perp\!\!\!\perp$.

Proof: By induction on typing derivations.

Normalization

If $\vdash_l c\tau$ then $c\tau$ normalizes.

Proof: The set $\perp\!\!\!\perp_{\Downarrow} = \{c\tau \in C_0 : c\tau \text{ normalizes}\}$ *is a pole.*

To sum up

Initial questions:

- ✓ Does typed terms normalize? *Yes!*
- ✓ Can we define a realizability interpretation? *Yes!*

Bonus:

- Scales to 2nd order types for free
- Seems to be a generic method for calculi with memory

To sum up

Initial questions:

✓ Does typed terms normalize? *Yes!*

✓ Can we define a realizability interpretation? *Yes!*

Bonus:

- Scales to 2nd order types for free
- Seems to be a generic method for calculi with memory

Continuation-and-environment passing style translations

Towards typed translations

Intuitions

(Analyzing Ariola *et al.* '12)

Sequent calculus:

$$\langle t \,\|\, e \rangle \tau$$

$$\underset{\text{Term}}{\nearrow} \qquad \underset{\text{Context}}{\uparrow} \qquad \underset{\text{Environment}}{\nwarrow}$$

Untyped CEPS:

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

$$\underset{\text{environment passing}}{\nearrow} \qquad \underset{\text{continuation passing}}{\nwarrow}$$

Intuitions

(Analyzing Ariola *et al.* '12)

Untyped CEPS:

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

environment passing — continuation passing

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

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)

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

environment passing

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)

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

environment passing — $[\![\tau]\!]_{\tau}$

continuation passing — $[\![t]\!]_{\mathrm{t}}$

Step 1 - Continuation-passing part

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

$$\begin{array}{rcl}
[\![\tilde{\mu}x.c]\!]_{\mathrm{e}} & = & \lambda t.[\![c]\!]_{\mathrm{c}} \\
[\![\mu\alpha.c]\!]_{\mathrm{t}} & = & \lambda\alpha.[\![c]\!]_{\mathrm{c}} \\
 & \vdots &
\end{array}$$

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

Typing the CEPS: guidelines (2/4)

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

Step 2- Environment-passing part

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

$$\downarrow$$

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

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

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

$$\downarrow$$

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

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

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

$$\downarrow$$

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

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

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

$$\downarrow$$

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

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

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

A possible reduction scheme:

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

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	$\rightarrow^* \langle V \parallel \tilde{\mu}[x].\langle x \parallel F\rangle \tau_2\rangle \tau_1 \boxed{\tau'}$

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

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:

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

Store subtyping:

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

Translation:

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

$$\downarrow$$

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

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:

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

$$\downarrow$$

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

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:

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

$$\downarrow$$

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

(reminiscent of Kripke forcing)

Typing the CEPS: guidelines (3/4)

Step 3 - Extension of the environment

Key idea:

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

Store subtyping:

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

Translation:

$$\begin{array}{rcl}
[\![\Gamma]\!] \triangleright_\mathsf{e} A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{t} A \to \bot \\
[\![\Gamma]\!] \triangleright_\mathsf{t} A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{E} A \to \bot \\
[\![\Gamma]\!] \triangleright_\mathsf{E} A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{V} A \to \bot \\
[\![\Gamma]\!] \triangleright_\mathsf{V} A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{F} A \to \bot \\
[\![\Gamma]\!] \triangleright_\mathsf{F} A & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{v} A \to \bot \\
[\![\Gamma]\!] \triangleright_\mathsf{v} A \to B & \triangleq & \forall \Upsilon <: [\![\Gamma]\!].\Upsilon \to \Upsilon \triangleright_\mathsf{t} A \to \Upsilon \triangleright_\mathsf{E} B \to \bot
\end{array}$$

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:

$$\frac{\Gamma(n) = (x_n : T)}{\Gamma \vdash_V x_n : T}$$

$$\langle x_n \| F\rangle\tau[x_n := t]\tau \xrightarrow{n=|\tau|} \langle t \| \tilde{\mu}[x_n].\langle x_n \| F\rangle\tau'\rangle\tau$$

$$\langle V \| \tilde{\mu}[x_i].\langle x_i \| F\rangle\tau'\rangle\tau \xrightarrow{n=|\tau|} \langle V \| \uparrow_i^n F\rangle\tau[x_n := V]\uparrow_i^n\tau'$$

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:

- and the target:

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

$$\downarrow$$

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

Typing the CEPS: guidelines (4/4)

Step 4 - Avoiding name clashes

Here, we use De Bruijn levels both:

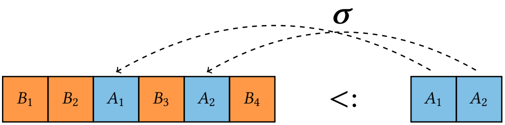
- and the target:

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

$$\downarrow$$

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

...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

A calculus of expandable stores
Herbelin & M. [2020]

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

A calculus of expandable stores
Herbelin & M. [2020]

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
   ↬ *Here, simply-typed calculi*
2. a syntax for **stores and coercions**
3. the **target calculus**, an instance of F_Υ

Principles

A calculus of expandable stores
Herbelin & M. [2020]

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

A calculus of expandable stores
Herbelin & M. [2020]

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 the 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 the 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 the 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:

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

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:

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

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

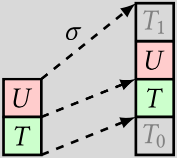
$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow \sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

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

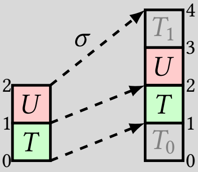
$$\frac{\Gamma \vdash \sigma : \Upsilon' <: \Upsilon}{\Gamma \vdash \Uparrow\sigma : (\Upsilon', F) <: \Upsilon}\,(<:_{\Uparrow})$$

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 +

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

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

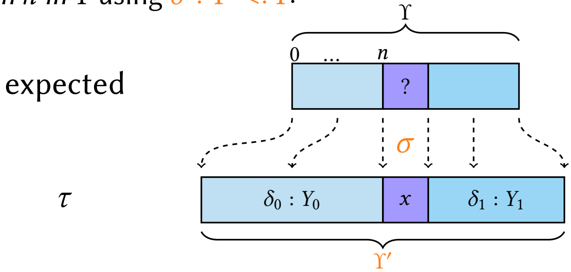
System F_Υ

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

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

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

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

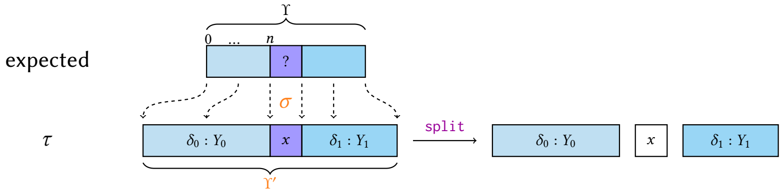


System F_Υ

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

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

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 +

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

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:

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

▶ *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:

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

▶ *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:

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:

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

▶ *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

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 ≅ Kripke forcing interleaved with a negative translation

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

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

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?

$$\boxed{\cdot \triangleright_t A \text{ is a function : store type} \mapsto \text{type}}$$

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?

$$\boxed{\cdot \triangleright_t A \text{ is a function : store type} \mapsto \text{type}}$$

$$\begin{array}{rcl} \square\mathcal{F} & \triangleq & \Upsilon \;\mapsto\; \forall \Upsilon' <: \Upsilon . \Upsilon' \to (\mathcal{F}\,\Upsilon') \to \bot \\ \cdot \triangleright_t A & = & \square(\cdot \triangleright_E A) = \square(\square(\cdot \triangleright_V A)) = \ldots \end{array}$$

Thank you for your attention.