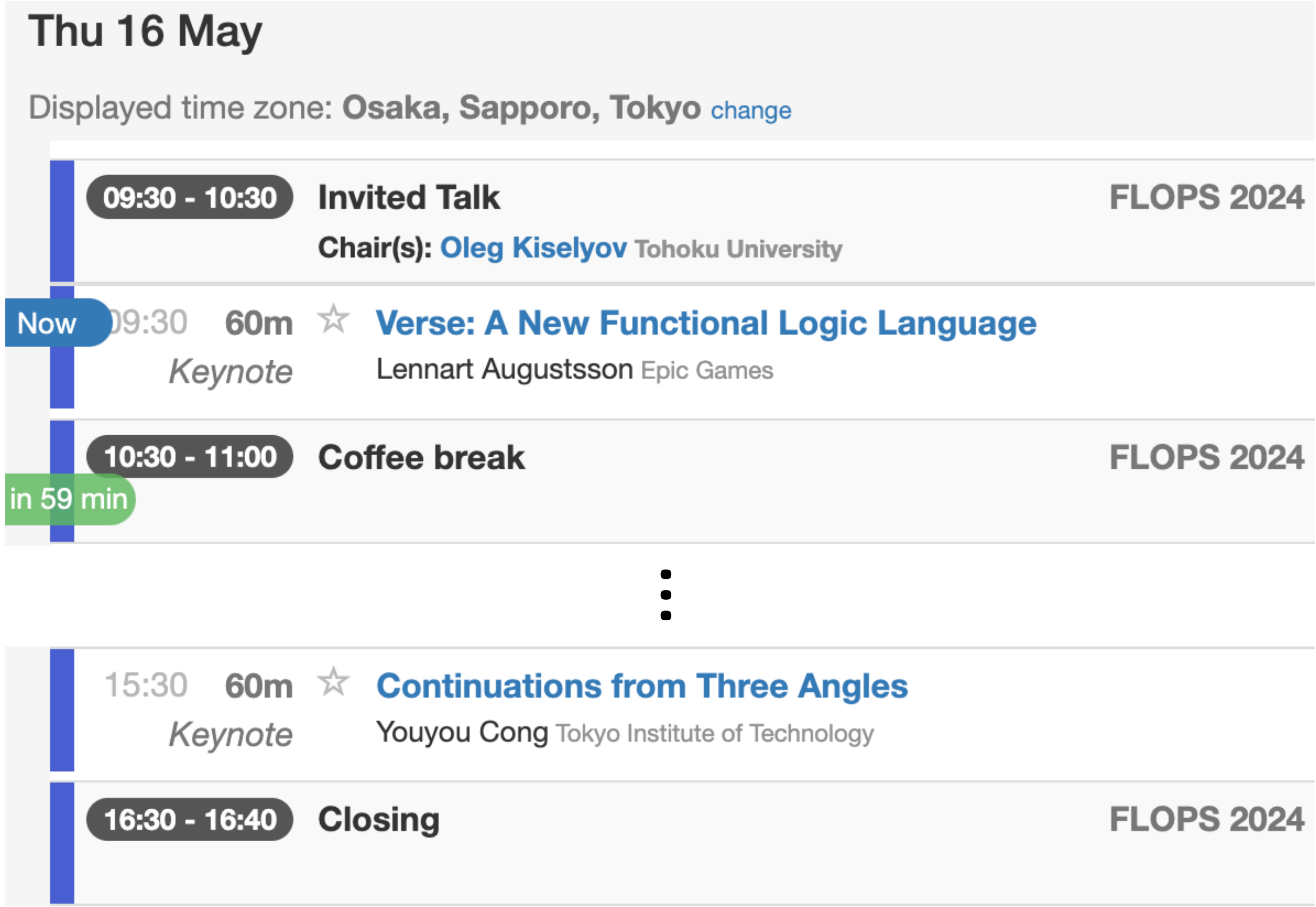
rest of computation

Continuations from Three Angles

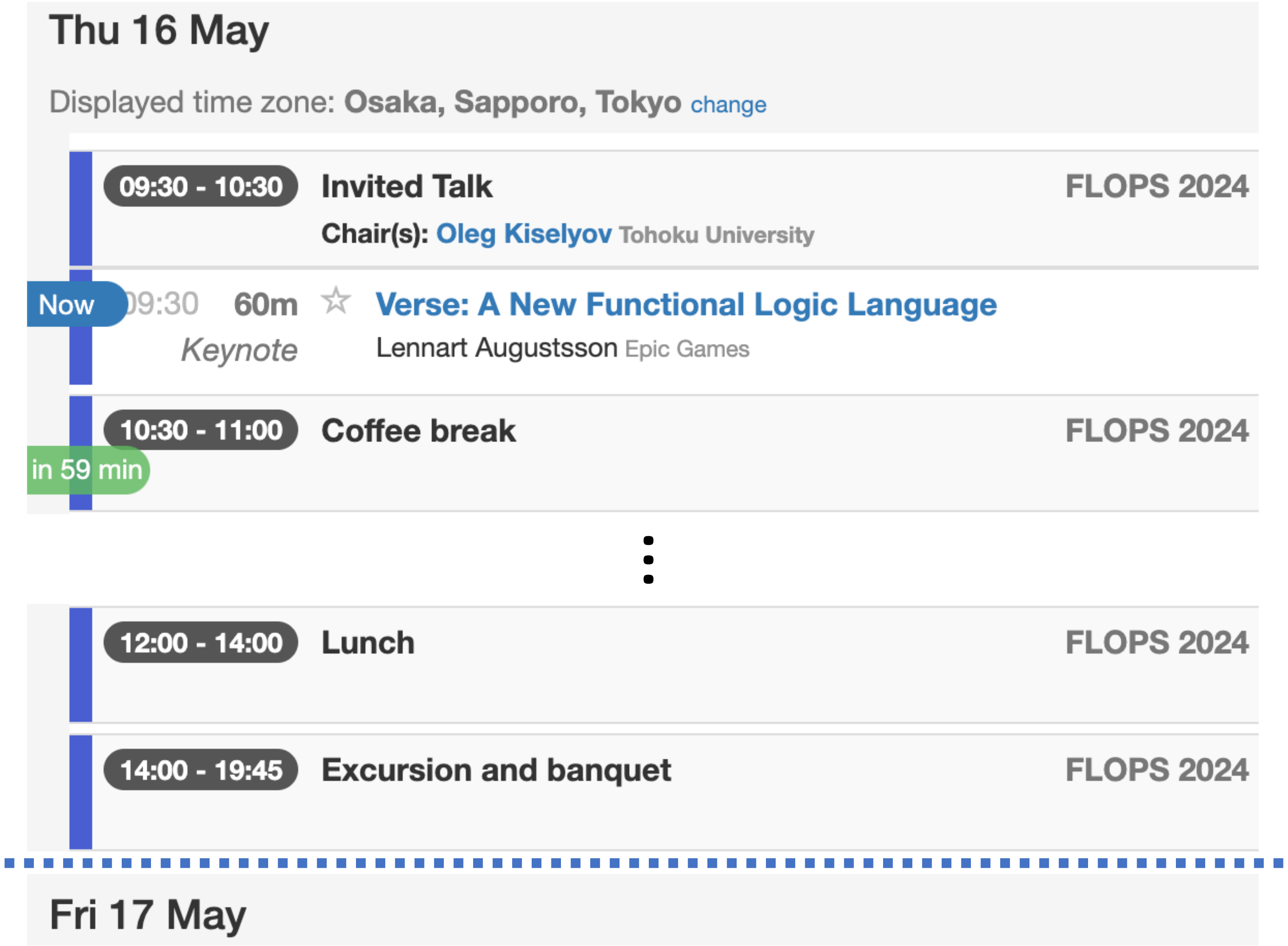
Youyou Cong (Tokyo Institute of Technology)

FLOPS 2024 Keynote

Continuations in real life



Delimited continuations in real life



Expressiveness of continuations

- Exceptions
- Nondeterminism
- State
- Generators/iterators
- Futures/promises
- Async/await

Implementations of continuations

Racket

OCaml

Haskell

Prolog

Koka

Links

Effekt

My continuations research

Applications	Theory	Learning

2013 2024

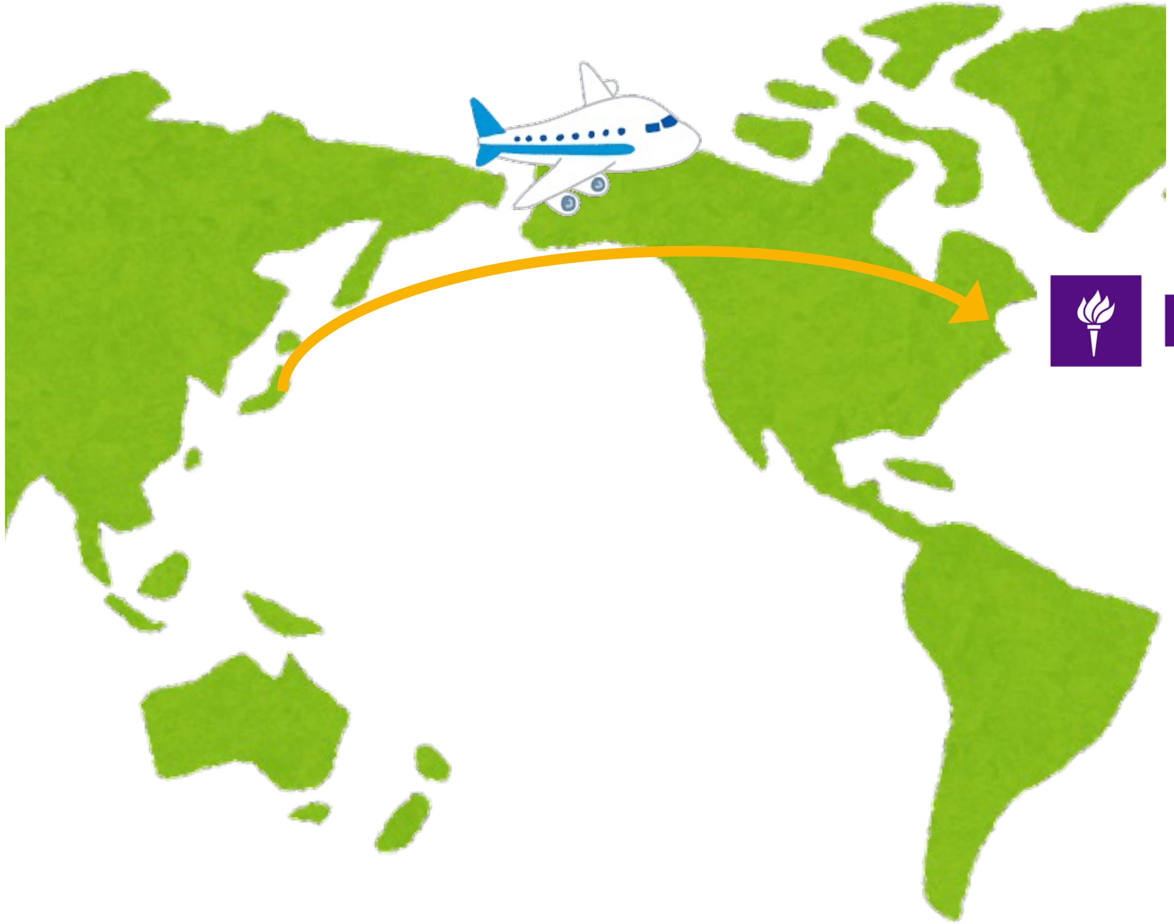
2013: Joining NL lab at Ochanomizu

Bekkilab Official Website

Give everyone on earth one source of truth for logic, language and information science.

戸次研究室では、数理言語学の研究をしています。数理言語学とは、数理モデルを用いて自然言語の構造を解明する学問です。主に、組合せ範疇文法(Combinatory Categorial Grammar)／依存型意味論(Dependent Type Theory)／圏論(Category theory)などを用いて言語現象（日本語の統語構造／照応・前提／談話関係／一般化量化子／モダリティ／慣習的含み／敬語／フォーカス／含意関係認識／日本語の機能表現／日本語のテンス・アスペクトなど）全体を統一的に説明する理論と、その背後にある構造を追究しています。

A research visit to NYU



NYU

Chris Barker

Calculating the meaning of sentences

John loves Mary → $Love(j, m)$

Calculating the meaning of sentences

John $\triangleleft$ (loves $\triangleright$ Mary) → $Love(j, m)$

John = j

Mary = m

loves = $\lambda o. \lambda s. Love(s, o)$

Quantification [Shan '04]

John loves everyone → $\forall x. Love(j, x)$

Quantification [Shan '04]

John $\lhd$ (loves $\rhd$ everyone) ⟶ $\forall x. Love(j, x)$

Quantification [Shan '04]

John ◁ (loves ▷ everyone) ⟶ $\forall x. Love(j, x)$

John	=	j
loves	=	$\lambda o. \lambda s. Love(s, o)$
everyone	=	??

Quantification [Shan '04]

John $\lhd$ (loves $\rhd$ everyone) $\longrightarrow$ $\forall x.\, Love(j, x)$

$$\begin{aligned} \text{John} &= j \\ \text{loves} &= \lambda o.\, \lambda s.\, Love(s, o) \\ \text{everyone} &= \boldsymbol{S}k.\, \forall x.\, k\ x \end{aligned}$$

shift

Quantification [Shan '04]

John ◁ (loves ▷ everyone) ⟶ $\forall x. Love(j, x)$

$$\langle E[Sk.e] \rangle \to \langle e[\lambda y. \langle E[y] \rangle / k] \rangle$$

reset

shift

Quantification [Shan '04]

$\langle \text{John} \lhd (\text{loves} \rhd \text{everyone}) \rangle$ ⟶ $\forall x. Love(j, x)$

John	=	j
loves	=	$\lambda o. \lambda s. Love(s, o)$
everyone	=	$Sk. \forall x. k\ x$

Deep vs. shallow semantics in PL

deep

(Danvy & Filinski's shift)

<E[Sk. e]>

↓

k = λy. <E[y]>

shallow

(Felleisen's control)

<E[Ck. e]>

↓

k = λy. E[y]

Deep vs. shallow semantics in NL [Cong+ '15]

John introduced someone to everyone

$\exists x. \forall y. Introduce(j, x, y)$

direct scope reading
(natural)

$\forall y. \exists x. Introduce(j, x, y)$

inverse scope reading
(hard)

Deep vs. shallow semantics in NL [Cong+ '15]

John introduced someone to everyone

$\exists x. \forall y. Introduce(j, x, y)$

$\forall y. \exists x. Introduce(j, x, y)$

someone = $\boldsymbol{S} k. \exists x. k\ x$

everyone = $\boldsymbol{S} k. \forall y. k\ y$

deep, i.e., continuation includes reset

Deep vs. shallow semantics in NL [Cong+ '15]

John introduced someone to everyone

$\exists x. \forall y. Introduce(j, x, y)$

$\forall y. \exists x. Introduce(j, x, y)$

shallow, i.e., continuation does not include reset

someone $=$ $\mathcal{C}k. \exists x. k\ x$

everyone $=$ $\mathcal{C}k. \forall y. k\ y$

Deep vs. shallow semantics in NL [Cong+ '15]

John introduced someone to everyone

$$\exists x. \forall y. Introduce(j, x, y) \qquad \forall y. \exists x. Introduce(j, x, y)$$

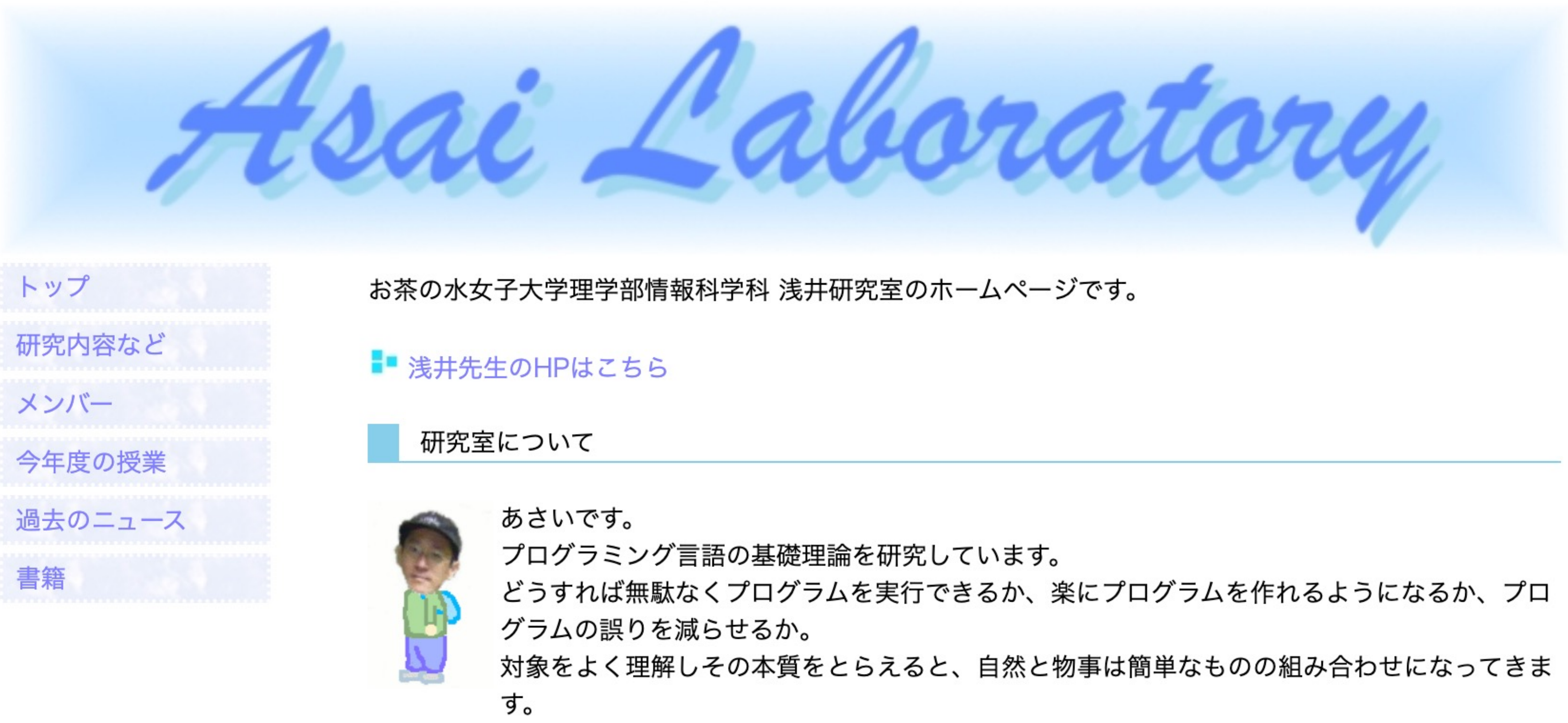
someone = $\boldsymbol{S}k. \exists x. k\ x$

everyone = $\boldsymbol{S}k. \forall y. k\ y$

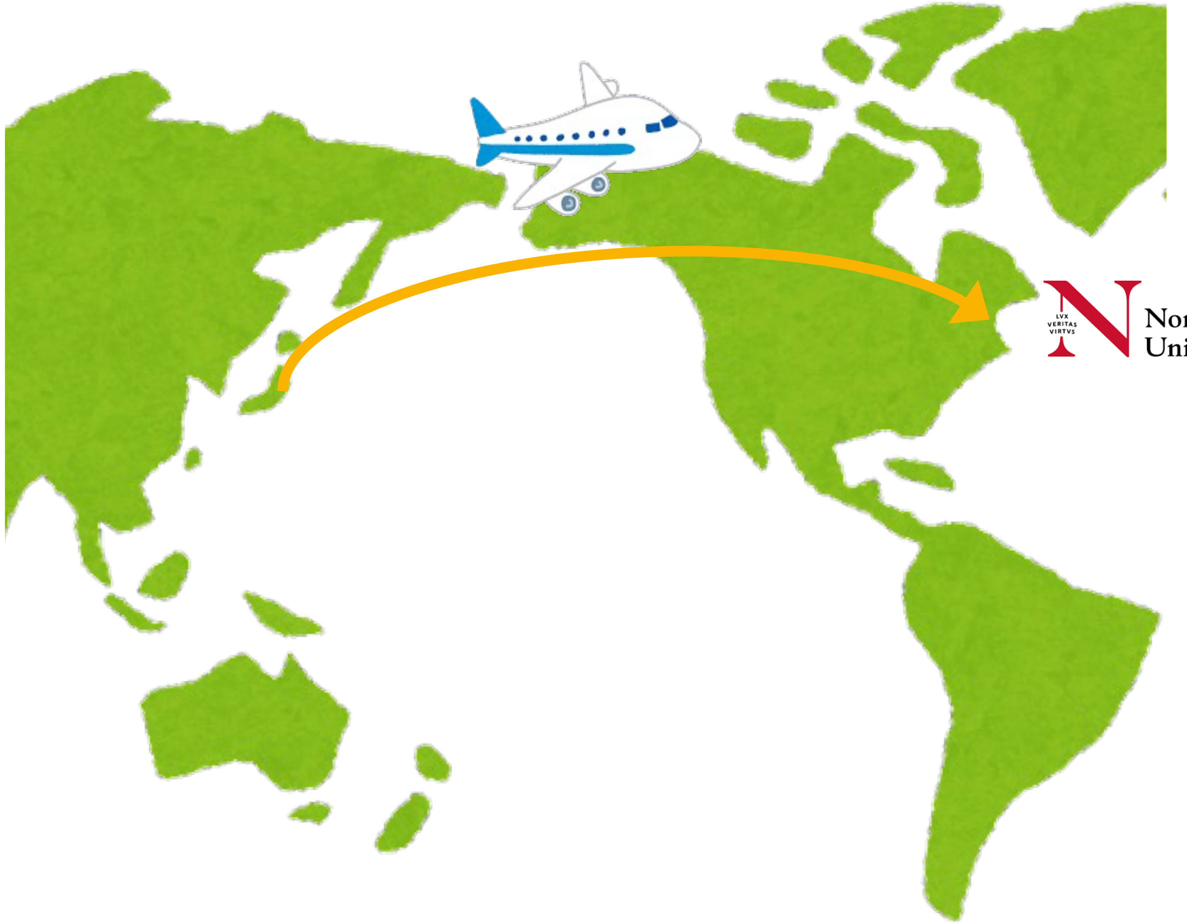
someone = $\boldsymbol{C}k. \exists x. k\ x$

everyone = $\boldsymbol{C}k. \forall y. k\ y$

2016: Joining PL lab at Ochanomizu



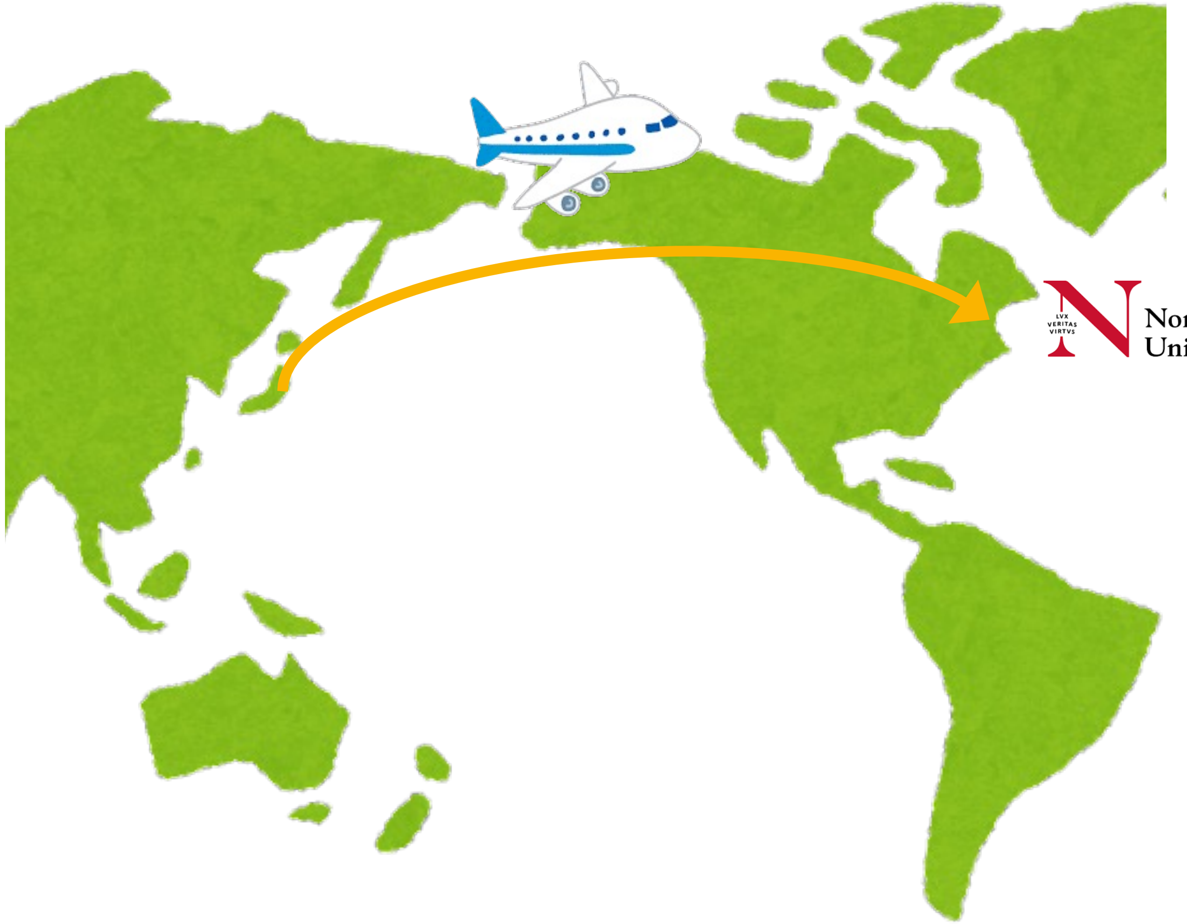
A research visit to NEU



Northeastern University

Matthias Felleisen

A research visit to NEU

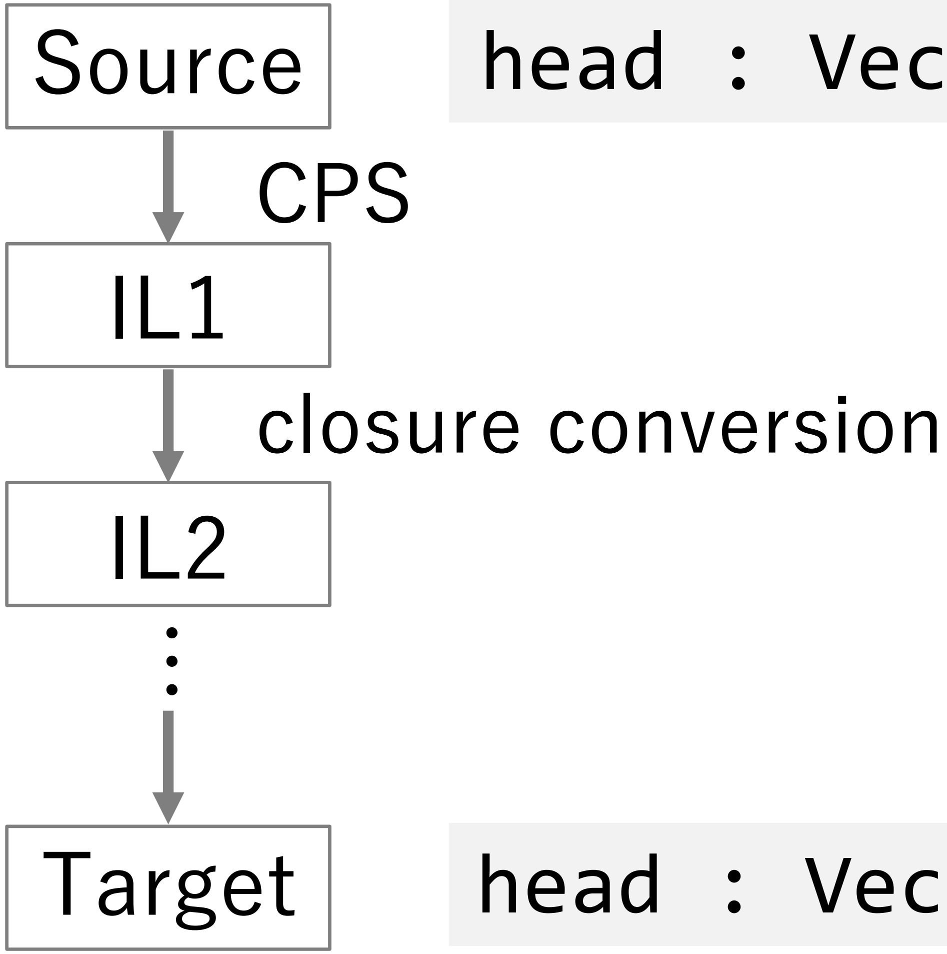


William Bowman

Nick Rioux

Amal Ahmed

Dependent type preserving compilation



`head : Vec A (suc n) → A`

`head : Vec ⟦A⟧ (suc n) → ⟦A⟧`

Known challenge

Barthe,
Hatcliff,
Sørensen
HOSC '99

> they [CPS translations] cannot be extended readily to Σ types as the usual translation for pairs does not preserve typing.

$$\frac{\Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : B[e_1/x]}{\Gamma \vdash (e_1,\ e_2) : \Sigma x : A.\ B}$$

$$\frac{\Gamma \vdash e : \Sigma x : A.\ B}{\Gamma \vdash \mathtt{snd}\ e : B[\mathtt{fst}\ e/x]}$$

CPS'ing second projection (simply typed)

`snd e : B` where `e : A × B`

↓

⟦B⟧ → ⊥

`λk. ⟦e⟧ (λv. snd v k) : (⟦B⟧ → ⊥) → ⊥`

(⟦B⟧ → ⊥) → ⊥

CPS'ing second projection (dependently typed)

$$\texttt{snd e : B[fst e / x]} \quad \texttt{where e : } \Sigma \texttt{x : A. B}$$

$$\llbracket \texttt{B} \rrbracket [\llbracket \texttt{fst e} \rrbracket / \texttt{x}] \rightarrow \bot$$

$$\lambda \texttt{k}. \llbracket \texttt{e} \rrbracket \; (\lambda \texttt{v. snd v k}) \; : \; (\llbracket \texttt{B} \rrbracket [\llbracket \texttt{fst v} \rrbracket / \texttt{x}] \rightarrow \bot) \rightarrow \bot$$

$$(\llbracket \texttt{B} \rrbracket [\texttt{fst v} / \texttt{x}] \rightarrow \bot) \rightarrow \bot$$

Intuition

```
snd e : B[fst e / x]    where e : Σx : A. B
```

unique if e is pure

```
λk. ⟦e⟧ (λv. snd v k) : (⟦B⟧[⟦fst v⟧ / x] → ⊥) → ⊥

fst v = fst val-of(⟦e⟧) = fst (⟦e⟧ id) = ⟦fst e⟧
```

Solution [Bowman, Cong, Rioux, Ahmed '17]

1. Polymorphic answer type

$$e : A$$

$$\downarrow$$

$$\lambda\alpha.\ \lambda k.\ e' : \Pi\alpha.(\llbracket A \rrbracket \to \alpha) \to \alpha$$

$$\lambda\alpha.\lambda k.\llbracket e \rrbracket\ \alpha\ (\lambda v.\mathrm{snd}\ v\ \alpha\ k) \ :\ \Pi\alpha.(\llbracket B \rrbracket[\llbracket \mathrm{fst}\ e \rrbracket / x] \to \alpha) \to \alpha$$

$$\mathrm{fst}\ v = \mathrm{fst}\ \mathrm{val\text{-}of}(\llbracket e \rrbracket) = \mathrm{fst}\ (\llbracket e \rrbracket\ C\ \mathrm{id}) = \llbracket \mathrm{fst}\ e \rrbracket$$

$$\text{where } C = \Sigma x : \llbracket A \rrbracket'.\ \llbracket B \rrbracket'$$

Solution [Bowman, Cong, Rioux, Ahmed '17]

2. New typing rule

$$\frac{\Gamma \vdash e_1 : \forall\alpha.(A \to \alpha) \to \alpha \qquad \Gamma,\ \mathbf{v = e_1\ A\ id} \vdash e_2 : B}{\Gamma \vdash e_1\ B\ (\lambda v.\ e_2) : B}$$

$$\lambda\alpha.\lambda k.[\![e]\!]\ \alpha\ (\lambda v.\mathrm{snd}\ v\ \alpha\ k)\ :\ \Pi\alpha.([\![B]\!][[\![\mathrm{fst}\ e]\!] / x] \to \alpha) \to \alpha$$

$$\mathbf{fst\ v = fst\ val\text{-}of([\![e]\!]) = fst\ ([\![e]\!]\ C\ id)} = [\![\mathrm{fst}\ e]\!]$$

$$\text{where } C = \Sigma x : [\![A]\!]'.\ [\![B]\!]'$$

Solution [Bowman, Cong, Rioux, Ahmed '17]

3. New equivalence rule

$$\frac{}{e\ B\ k \equiv k\ (e\ A\ \mathrm{id})}$$

λα.λk.⟦e⟧ α (λv.snd v α k) : Πα.(⟦B⟧[⟦fst e⟧ / x] → α) → α

fst v = fst val-of(⟦e⟧) = **fst (⟦e⟧ C id) = ⟦fst e⟧**

where C = Σx : ⟦A⟧'. ⟦B⟧'

Type preservation

If $\Gamma \vdash e : A$ in the source,

then $[\![\Gamma]\!] \vdash [\![e]\!] : \Pi\alpha.\ ([\![A]\!] \to \alpha) \to \alpha$ in the target.

Proof: By induction on the typing derivation.

Known challenge

Herbelin
TLCA '05

> We show that a minimal dependent type theory based on Σ-types and equality is degenerated in presence of computational classical logic.

CPS of second projection (pure)

snd e : B[fst e / x]   where e : Σx : A. B

$\downarrow$

unique if e is pure

λα.λk.⟦e⟧ α (λ**v**.snd v α k) : Πα.(⟦B⟧[⟦fst e⟧ / x] → α) → α

fst v = fst val-of(⟦e⟧) = fst (⟦e⟧ C id) = ⟦fst e⟧

where C = Σx : ⟦A⟧'. ⟦B⟧'

CPS of second projection (impure)

$\mathtt{snd\ e} : \mathtt{B[fst\ e / x]}$ where $\mathtt{e} : \Sigma \mathtt{x} : \mathtt{A.\ B}$

↓

not unique if e is impure
(e.g., $\mathtt{e} = \mathcal{S}\mathtt{k.\ k\ 1 + k\ 2} \Rightarrow \mathtt{v} = 1, 2$)

$\lambda\alpha.\lambda \mathtt{k}.[\![\mathtt{e}]\!]\ \alpha\ (\lambda \mathbf{v}.\mathtt{snd\ v}\ \alpha\ \mathtt{k}) : \Pi\alpha.([\![\mathtt{B}]\!][[\![\mathtt{fst\ e}]\!] / \mathtt{x}] \to \alpha) \to \alpha$

$\mathtt{fst\ v} \neq \mathtt{fst\ val\text{-}of}([\![\mathtt{e}]\!]) = \mathtt{fst}\ ([\![\mathtt{e}]\!]\ \mathtt{C\ id}) = [\![\mathtt{fst\ e}]\!]$

where $\mathtt{C} = \Sigma \mathtt{x} : [\![\mathtt{A}]\!]'.\ [\![\mathtt{B}]\!]'$

Solution: purity restriction [Cong & Asai '18]

Types may depend only on pure terms

✓ `Vec Nat 3`

✗ `Vec Nat Sk.3`

Solution: purity restriction [Cong & Asai '18]

pure

$\Gamma \vdash_{p} e : A$

impure (change context type from α to β)

$\Gamma \vdash_{\alpha,\beta} e : A$

Solution: purity restriction [Cong & Asai '18]

e pure ⇒
may appear in types

$$\frac{\Gamma \vdash_{p} e : \Sigma x : A.\, B}{\Gamma \vdash_{p} \mathtt{snd}\ e : B[\mathtt{fst}\ e / x]}$$

e impure ⇒
may not appear in types

$$\frac{\Gamma \vdash_{\alpha,\beta} e : A \times B}{\Gamma \vdash_{\alpha,\beta} \mathtt{snd}\ e : B}$$

Type preservation

- If $\Gamma \vdash_p e : A$ in the source,
  then $[\![\Gamma]\!] \vdash [\![e]\!] : \Pi\alpha.\,([\![A]\!] \to \alpha) \to \alpha$ in the target.
- If $\Gamma \vdash_{\alpha,\beta} e : A$ in the source,
  then $[\![\Gamma]\!] \vdash [\![e]\!] : ([\![A]\!] \to [\![\alpha]\!]) \to [\![\beta]\!]$ in the target.

Proof: By induction on the typing derivation.

2019: Joining PRG at Tokyo Tech

Programming Research Group

Department of Mathematical and Computing Science,
Tokyo Institute of Technology

A visitor from Linköping

LINKÖPING UNIVERSITY

Filip Strömbäck

A language for learning continuations

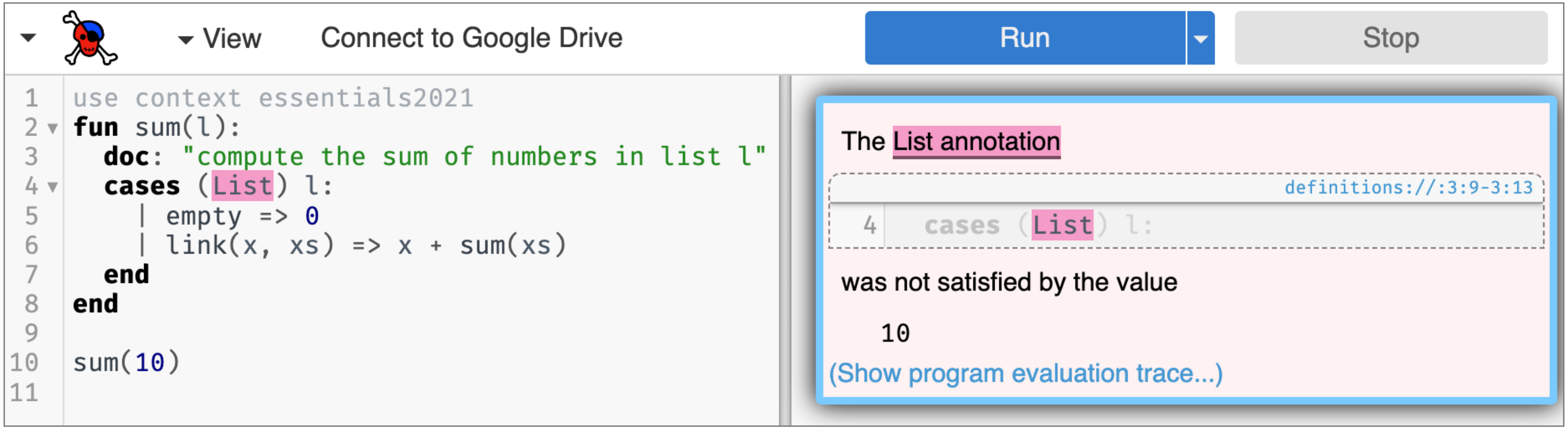
Pyret +

- ☑ control constructs
- ☑ stepping evaluator
- ☐ continuation visualizer

Example of Pyret program

```
fun sum(l):
  doc: "compute the sum of numbers in list l"
  cases (List) l:
    | empty => 0
    | link(x, xs) => x + sum(xs)
  end
end
```

Pyret online editor



Extension 1: grab/delimit

```
delimit:   [reset]
  2 * grab(k): k(k(5)) end
end  # 20  [shift]
```

Cf: lambda abstraction

```
lam(x): 2 * x end
```

Extension 2: send/run

```
run:
  2 * send(5)
with handler(x, k): k(k(x))
end  # 20
```

fcontrol

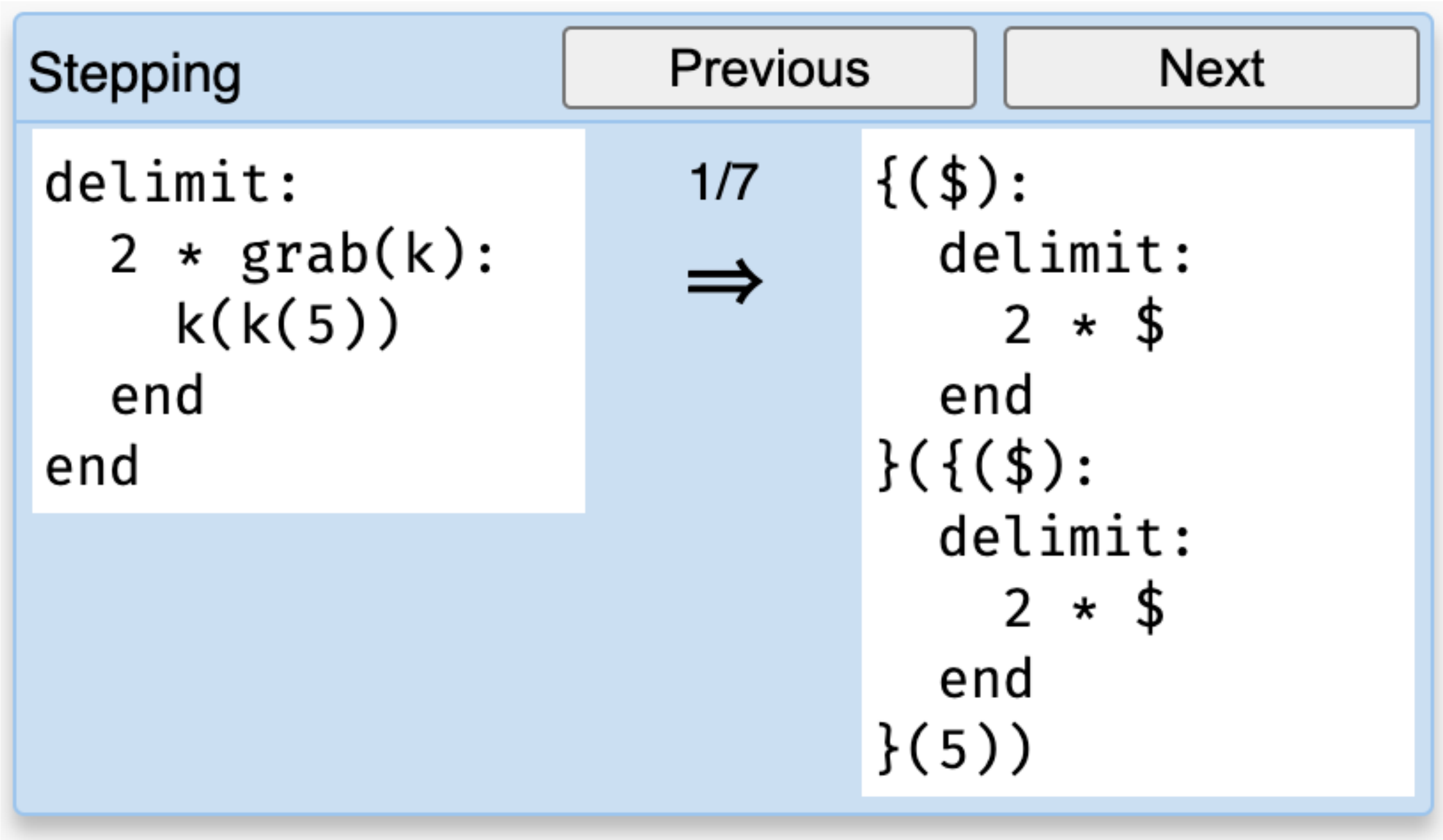
Extension 3: effect handlers

```
effect Exn:
  | raise(v)
end

handler (Exn) abort:
  | raise(v), k => v
end


handle: 2 * raise(5) with abort  # 5
```

Stepping evaluator



A course on continuations (June – July)

1. Introduction
2. FP basics
3. CPS
4. grab/delimit
5. send/run
6. Effect handlers
7. Mixing effects
8. Linguistic applications
9. Untyped λ calculus
10. Typed λ calculus
11. λ + grab/delimit
12. λ + send/run
13. λ + effect handlers
14. Wrap-up

Continuation of my research

- Applications:

  Reactive programming, probabilistic programming

- Theory:

  Compilation of non-algebraic effects

- Learning:

  Students' difficulties, learning support

Wrapping up

Continuations are interesting to study!

Thanks to...