אח

Bartłomiej Królikowski
presenting work by Dariusz Biernacki, Bartłomiej Królikowski and Piotr Polesiuk

27 June 2025

Goal

- CC with algebraic effects

Goal

- CC with algebraic effects
- for now: CC with call/cc (CC for classical logic)

Why algebraic effects?

(together with handlers)

- they are composable (unlike monads)

Why algebraic effects?

(together with handlers)

- they are composable (unlike monads)
- they provide separation of interface and implementation of effects

Why algebraic effects?

(together with handlers)

- they are composable (unlike monads)
- they provide separation of interface and implementation of effects
- they allow users to choose a particular implementation of effects presented by a program

What do we want?

CC with effects tracked by a type-and-effect system

Problems

- effects in type expressions?

Problems

- effects in type expressions?
- dependent functions applied to impure arguments?

Problems

- effects in type expressions?
- dependent functions applied to impure arguments?
- type preserving CPS translation of dependent types is tricky

Problems

- effects in type expressions?
- dependent functions applied to impure arguments?
- type preserving CPS translation of dependent types is tricky
- we want to have subtyping (see: Materzok and Biernacki 2011) - harder in presence of dependent types (Chen 1997)

Solutions

- well-formed types have to be pure

(just like in Cong 2019)

Solutions

- well-formed types have to be pure
- no dependency on impure arguments

(just like in Cong 2019)

Solutions

- well-formed types have to be pure
- no dependency on impure arguments
- selective translation (i.e. only for impure terms)

(just like in Cong 2019)

How to prevent unwanted impurity?

- syntactic restrictions (see: Herbelin 2012)

How to prevent unwanted impurity?

- syntactic restrictions (see: Herbelin 2012)
- type-and-effect system (see: Cong 2019) ←—

How to prevent unwanted impurity?

- syntactic restrictions (see: Herbelin 2012)
- type-and-effect system (see: Cong 2019) ⟵

  . . . but what is the logical meaning of effect types?

A simpler example: `call/cc`

Classical logic via Peirce's law (`call/cc`):

$$\frac{\Gamma, k : \neg\tau \vdash e : \tau}{\Gamma \vdash \mathcal{K}k.e : \tau}$$

the rule of contradiciton (`throw`):

$$\frac{\Gamma \vdash e_1 : \neg\tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \lhd e_2 : \tau'}$$

Alternative presentations

if we want to avoid the empty type ($\bot$) or negation:

- $\lambda\mu$ (Parigot 1992) style presentation:

$$\frac{\Gamma \vdash e : \tau | \Delta, \alpha : \tau}{\Gamma \vdash \mathcal{K}\alpha.e : \tau | \Delta} \qquad \frac{(\alpha : \tau) \in \Delta \qquad \Gamma \vdash e : \tau | \Delta}{\Gamma \vdash \alpha \lhd e : \tau' | \Delta}$$

Alternative presentations

if we want to avoid the empty type ($\bot$) or negation:

- $\lambda\mu$ (Parigot 1992) style presentation:

$$\frac{\Gamma \vdash e : \tau | \Delta, \alpha : \tau}{\Gamma \vdash \mathcal{K}\alpha.e : \tau | \Delta} \qquad \frac{(\alpha : \tau) \in \Delta \qquad \Gamma \vdash e : \tau | \Delta}{\Gamma \vdash \alpha \lhd e : \tau' | \Delta}$$

- Wadler-style symmetric calculus (Wadler 2003)

Alternative presentations

if we want to avoid the empty type ($\bot$) or negation:

- $\lambda\mu$ (Parigot 1992) style presentation:

$$\frac{\Gamma \vdash e : \tau | \Delta, \alpha : \tau}{\Gamma \vdash \mathcal{K}\alpha.e : \tau | \Delta} \qquad \frac{(\alpha : \tau) \in \Delta \qquad \Gamma \vdash e : \tau | \Delta}{\Gamma \vdash \alpha \lhd e : \tau' | \Delta}$$

- Wadler-style symmetric calculus (Wadler 2003)
- polarized set of variables: positive x and negative $\alpha \longleftarrow$

$$\frac{\Gamma, \alpha : \tau \vdash e : \tau}{\Gamma \vdash \mathcal{K}\alpha.e : \tau} \qquad \frac{(\alpha : \tau) \in \Gamma \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \alpha \lhd e : \tau'}$$

Effect type = logic

- effect types: $\varepsilon ::= p \mid i$

Effect type = logic

- effect types: $\varepsilon ::= p \mid i$
- p - pure expression (constructive logic)

Effect type = logic

- effect types: $\varepsilon ::= p \mid i$
- p - pure expression (constructive logic)
- i - impure expression (classical logic)

Effect type = logic

- effect types: $\varepsilon ::= p \mid i$
- p - pure expression (constructive logic)
- i - impure expression (classical logic)
- "subeffecting": $p <: i$
  (everything true in constructive logic is true in classical logic)

Example rules

$$\frac{\Gamma \vdash A : */p \qquad \Gamma, \alpha : A \vdash e : A/i}{\Gamma \vdash \mathcal{K}\alpha : A.e : A/i}$$

$$\frac{\Gamma \vdash \Pi x : A.B : */p \qquad \Gamma \vdash e_1 : \Pi^{\varepsilon} x : A.B/\varepsilon \qquad \Gamma \vdash e_2 : A/p}{\Gamma \vdash e_1\ e_2 : [e_2/x]B/\varepsilon}$$

$$\frac{\Gamma \vdash A : */p \quad \Gamma \vdash B : */p \qquad \Gamma \vdash e_1 : A \to_{\varepsilon} B/\varepsilon \qquad \Gamma \vdash e_2 : A/\varepsilon}{\Gamma \vdash e_1\ e_2 : B/\varepsilon}$$

$$\frac{\Gamma \vdash A : */\varepsilon \qquad \Gamma, x : A \vdash B : */\varepsilon}{\Gamma \vdash \Pi^{\varepsilon} x : A.B : */\varepsilon}$$

Subtyping

- example rule (?):

$$\frac{\Gamma \vdash A' <: A : * \qquad \Gamma, x : A \vdash B <: B' : * \qquad \varepsilon <: \varepsilon'}{\Gamma \vdash \Pi^{\varepsilon} x : A.B <: \Pi^{\varepsilon'} x : A.B : *}$$

Subtyping

- example rule (?):

$$\frac{\Gamma \vdash A' <: A : * \qquad \Gamma, x : A \vdash B <: B' : * \qquad \varepsilon <: \varepsilon'}{\Gamma \vdash \Pi^{\varepsilon} x : A.B <: \Pi^{\varepsilon'} x : A.B : *}$$

- Work in progress

Reduction

- $\Gamma \vdash e_1 \longrightarrow e_2 : A/\varepsilon$

Reduction

- $\Gamma \vdash e_1 \longrightarrow e_2 : A/\varepsilon$
- example rule:

$$\frac{\Gamma \vdash e_1 : A \rightarrow_i B/i \qquad \Gamma \vdash e_2 : A/i}{\Gamma \vdash e_1\ (\mathcal{K}\alpha : A.e_2) \rightarrow_i \mathcal{K}\beta : B.e_1\ (e_2\ \{^{e_1\ \beta}/_{\alpha}\}) : B}$$

Reduction

- $\Gamma \vdash e_1 \longrightarrow e_2 : A/\varepsilon$
- example rule:

$$\frac{\Gamma \vdash e_1 : A \to_i B/i \qquad \Gamma \vdash e_2 : A/i}{\Gamma \vdash e_1\ (\mathcal{K}\alpha : A.e_2) \to_i \mathcal{K}\beta : B.e_1\ (e_2\ \{^{e_1\ \beta}/_{\alpha}\}) : B}$$

- Work in progress

Summary

We (are trying to) combine following features in our language:

- CC (dependent types + polymorphism + type constructors)
- Subtyping
- (a variant of) call/cc - classical logic
- effect types

We plan to extend it to algebraic effects.

Why CPS for dependent types is hard?

$$e_1\ e_2, \text{ where } (e_1 : \Pi x : A.B) \text{ and } (e_2 : A)$$

CPS:

$$\lambda k.\, [\![e_1]\!]\ (\lambda f.e_2\ (\lambda x.f\ x\ k))$$

CPS (Church style):

$$\lambda k : [?/x]\,[\![B]\!]^T \to O.$$
$$[\![e_1]\!]\ (\lambda f : (\Pi x : ([\![A]\!]^T.([\![B]\!]^T \to O) \to O)).$$
$$[\![e_2]\!]\ (\lambda x : [\![A]\!].f\ x\ k))$$

(see: Bowman et al. 2017)

References I

- Bowman, William J. et al. (Dec. 2017). "Type-preserving CPS translation of Sigma and Pi types is not not possible". In: *Proc. ACM Program. Lang.* 2.POPL. DOI: 10.1145/3158110. URL: https://doi.org/10.1145/3158110.
- Chen, Gang (1997). "Subtyping Calculus of Construction (Extended Abstract)". In: *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*. MFCS '97. Berlin, Heidelberg: Springer-Verlag, 189–198. ISBN: 3540634371.
- Cong, Youyou (2019). "Abstracting Control with Dependent Types". Graduate School of Humanities and Sciences, Advanced Sciences. PhD thesis. OCHANOMIZU UNIVERSITY.

References II


Herbelin, Hugo (2012). "A Constructive Proof of Dependent Choice, Compatible with Classical Logic". In: *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pp. 365–374. DOI: 10.1109/LICS.2012.47.


Materzok, Marek and Dariusz Biernacki (Sept. 2011). "Subtyping delimited continuations". In: *SIGPLAN Not.* 46.9, 81–93. ISSN: 0362-1340. DOI: 10.1145/2034574.2034786. URL: https://doi.org/10.1145/2034574.2034786.


Parigot, Michel (1992). "$\lambda\mu$-Calculus: An algorithmic interpretation of classical natural deduction". In: *Logic Programming and Automated Reasoning*. Ed. by Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 190–201. ISBN: 978-3-540-47279-7.

References III


Wadler, Philip (Aug. 2003). "Call-by-value is dual to call-by-name". In: *SIGPLAN Not.* 38.9, 189–201. ISSN: 0362-1340. DOI: 10.1145/944746.944723. URL: https://doi.org/10.1145/944746.944723.