

# Compositional Semantics for Composable Continuations

From Abortive to Delimited Control

Paul Downen    Zena M. Ariola

University of Oregon

ICFP'14 — September 1, 2014

# The big picture

- ▶ Effects that manipulate control flow, compositionally
  - ▶ Programs can refer to their context, but ...
  - ▶ Still have local, equational reasoning inside open programs
- ▶ Logic is an inspiration, ...
  - ▶ Lessons from logic can fix problems in programming
- ▶ Even with an untyped mindset
  - ▶ Sometimes, being type-agnostic is liberating!

# Classical control

- ▶ callcc is the **classic** control operator, going back to Scheme
- ▶ Classical control corresponds to **classical logic** (Griffin, 1990)
- ▶ Start with pure language, add primitive operations
  - ▶ Start with intuitionistic logic, add classical axioms
- ▶ Start with a language with continuation variables
  - ▶ Start with a logic with multiple conclusions

# Delimited control

- ▶ Delimit the scope of effects
- ▶ Continuations compose like functions
- ▶ Vastly more expressive power than classical control
  - ▶ Every monadic effect is simulated by delimited control (Filinski, 1994)
  - ▶ Exposes “monadic plumbing” underlying CBV languages

# Roadmap from classical to delimited control

Classical

$\lambda + \text{callcc}$



$\lambda\mu$

# Roadmap from classical to delimited control

Classical

$\lambda + \text{callcc}$

||

$\lambda\mu$

*syntactic  
relaxation*

$\Lambda\mu$

# Roadmap from classical to delimited control

Classical

Delimited

$\lambda + \text{callcc}$

$\lambda + \text{shift}_0 + \text{reset}_0$

||

||

$\lambda\mu$

$\Lambda\mu$

*syntactic  
relaxation*

# Classical control



# Operational semantics of callcc

- ▶ Extension of CBV  $\lambda$ -calculus

$$V ::= x \mid \lambda x.M$$
$$\mid \text{callcc}$$
$$\mid [E]$$

built-in function

reified evaluation context

$$M, N ::= V \mid M N$$
$$E ::= \square \mid E M \mid V E$$
$$E[(\lambda x.M) V] \mapsto E[M \{V/x\}]$$
$$E[\text{callcc } V] \mapsto E[V [E]]$$
$$E[[E'] V] \mapsto E'[V]$$

## Equational theory for callcc

- ▶ Reason more generally about open programs
- ▶ Extension of  $\lambda_c$  (Moggi, 1989)

$$\begin{array}{ll} \beta_v & (\lambda x.M) V = M\{V/x\} \\ \eta_v & \lambda x.V \ x = V \\ \beta_\Omega & (\lambda x.E[x]) M = E[M] \end{array}$$

- ▶ Add axioms that explain behavior of built-in callcc function (Sabry and Felleisen, 1993; Sabry, 1996)

## Problems of non-compositionality

- ▶ Equational theory weaker than operational semantics!
- ▶ Some programs can be evaluated to a value. . .

$$\text{callcc}(\lambda k. \lambda x. k (\lambda \_ . x)) \mapsto (\lambda x. [\square] (\lambda \_ . x))$$

- ▶ But the equational theory for callcc cannot reach a value!

$$\text{callcc}(\lambda k. \lambda x. k (\lambda \_ . x)) \neq V$$

- ▶ How can we know that we have the “whole” context?

## Of jumps and the extent of a continuation

- ▶ Calling a continuation never returns — it “jumps”
  - ▶  $E[[E'] 1]$  “jumps” out of  $E$  to  $E'$
  - ▶ Add variables  $\alpha, \beta, \dots$  that stand for continuations
  - ▶ Applying a continuation (variable) “jumps” (a.k.a. “aborts”)
- ▶ A jump  $\alpha M$  is the **same** when inside a larger evaluation context

$$E[\alpha M] = \alpha M \quad E \text{ is garbage}$$

- ▶ A jump **delimits** the usable extent of a continuation

## A running jump

- ▶ Let's try that again
- ▶ We can evaluate a jump to an answer...

$$\alpha (\text{callcc}(\lambda k.\lambda x.k (\lambda_.x))) \mapsto \alpha (\lambda x.[\alpha \square] (\lambda_.x))$$

- ▶ And the equational theory for callcc reaches that answer!

$$\alpha (\text{callcc}(\lambda k.\lambda x.k (\lambda_.x))) = \alpha (\lambda x.\alpha (\lambda_.x))$$

## $\lambda\mu$ : taking jumps seriously

- ▶ Syntactically distinguish jumps as “commands”

$$\begin{aligned} M, N &::= \dots \mid \mu\alpha.c && \text{control abstraction} \\ c &::= [\alpha]M && \text{command, a.k.a “jump”} \end{aligned}$$

- ▶ Commands “run”

$$\begin{aligned} [\alpha](E[(\lambda x.M)V]) &\mapsto [\alpha](E[M \{V/x\}]) \\ [\alpha](E[\mu\beta.c]) &\mapsto c \{[\alpha](E[N])/[\beta]N\} \end{aligned}$$

## $\lambda\mu$ : a language of classical logic

- ▶ Developed as calculus for classical logic (Parigot, 1992)
- ▶ Originally CBN, but also CBV (extension of  $\lambda_c$ ):

$$\begin{array}{l} \mu_E \quad [\alpha](E[\mu\beta.c]) = c \{[\alpha](E[N])/[\beta]N\} \\ \eta_\mu \quad \mu\alpha.[\alpha]M = M \\ \beta_\mu \quad (\lambda x.\mu\alpha.[\beta]M) N = \mu\alpha.[\beta]((\lambda x.M) N) \end{array}$$

- ▶ Equational theory contains operational semantics
- ▶  $\lambda\mu \equiv \lambda + \text{callcc!}$

# Relaxing the syntax



## $\Lambda\mu$ : a more relaxed language

- ▶ Collapse term/command distinction:  $M \equiv c$

$$M ::= \dots \mid \mu\alpha.M \mid [\alpha]M$$

- ▶ Same rules, just more expressive meta-variables:

$$\begin{array}{ll} (\lambda x.[\alpha]x) 1 = [\alpha]1 & \text{because } [\alpha]x \text{ is now a term} \\ [\alpha](\mu\_.1) = 1 & \text{because } 1 \text{ is now a command} \end{array}$$

## Nothing new, nothing gained?

- ▶ We haven't added any new constructs
- ▶ We haven't added any new rules
- ▶ As typed calculus,  $\Lambda\mu$  considered equivalent to Parigot's  $\lambda\mu$
- ▶ So they're the same?

## Nothing new, nothing gained?

- ▶ We haven't added any new constructs
- ▶ We haven't added any new rules
- ▶ As typed calculus,  $\Lambda\mu$  considered equivalent to Parigot's  $\lambda\mu$
- ▶ So they're the same? **No!**

# Delimited control

## shift and reset

- ▶ shift and reset are a common basis for delimited control

$$\text{reset}(E[\text{shift } V]) = \text{reset}(V (\lambda x.\text{reset}(E[x])))$$

- ▶ Continuations **return**, they are **composable** like normal functions

$$\begin{aligned} & 2 \times \text{reset}(10 + (\text{shift}(\lambda k.k (k 2)))) \\ &= 2 \times \text{reset}(10 + \text{reset}(10 + \text{reset}(2))) \\ &= 2 \times \text{reset}(22) = 44 \end{aligned}$$

$$\lambda + \text{shift} + \text{reset} \leq \Lambda\mu$$

- ▶ Embedding of shift and reset into  $\Lambda\mu$ 
  - ▶ Equational theory of shift and reset (Kameyama and Hasegawa, 2003) provable in  $\Lambda\mu$
  - ▶ The two-pass CPS transformation for shift and reset (Danvy and Filinski, 1990) derived from embedding
- ▶ So  $\lambda + \text{shift} + \text{reset}$  is a **subset** of  $\Lambda\mu$

$$\mu\alpha_1.\mu\alpha_2.\mu\alpha_3.4$$

$$[\alpha_3][\alpha_2][\alpha_1](f\ 0)$$

- ▶ What covers the whole of  $\Lambda\mu$ ?

## shift<sub>0</sub> and reset<sub>0</sub>

- ▶ Like shift, except that shift<sub>0</sub> **removes** its surrounding delimiter

$$\begin{aligned}\text{reset}(E[\text{shift } V]) &= \text{reset}(V (\lambda x.\text{reset}(E[x]))) \\ \text{reset}_0(E[\text{shift}_0 V]) &= V (\lambda x.\text{reset}_0(E[x]))\end{aligned}$$

- ▶ Many shift<sub>0</sub>s can “dig” out of many reset<sub>0</sub>s

$$\lambda + \text{shift}_0 + \text{reset}_0 \equiv \Lambda\mu$$

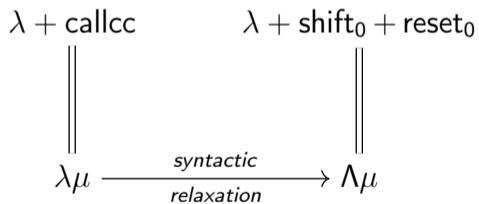
- ▶  $\lambda$  with  $\text{shift}_0$  and  $\text{reset}_0$  is **equivalent** to  $\Lambda\mu$ 
  - ▶ Equational theories correspond
  - ▶ CPS transforms correspond
  - ▶  $\text{shift}_0$  and  $\text{reset}_0$  rely on mixing terms with commands
- ▶ Restricting then relaxing the syntax led us from classical to delimited control!



# Roadmap from classical to delimited control

Classical

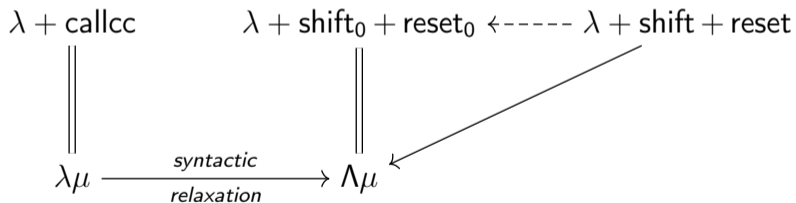
Delimited



# Roadmap from classical to delimited control

Classical

Delimited



## $\Lambda\mu$ : a framework for delimited control

- ▶ Encode both  $\text{shift}$ ,  $\text{reset}$  and  $\text{shift}_0$ ,  $\text{reset}_0$  in  $\Lambda\mu$
- ▶ Provable observational guarantees about the operators
  - ▶ Example: idempotency of  $\text{reset}$

$$\text{reset}(\text{reset}(M)) = \text{reset}(M)$$

- ▶ Observational guarantees still hold under composition
  - ▶  $\text{reset}$  is still idempotent even if we use  $\text{shift}_0$
  - ▶ Safely put together programs using either operators

## More in the paper

- ▶ Parameterize equational theory by different evaluation strategies
  - ▶ call-by-value, call-by-name, and call-by-need
- ▶ Improved reasoning for control operators in  $\lambda$ -calculus using continuation variables
- ▶ Equational correspondence with compositional transformations
  - ▶ Compositionality and hygiene makes life easier!

## Final words

- ▶ Control-flow effects: have our cake and eat it too
  - ▶ Expressive capability
  - ▶ Preserve local, open, high-level reasoning
  - ▶ Generic (parametric) treatment of evaluation strategies
- ▶ Compositionality is powerful
- ▶ Logic can be a wonderful guide

## References I

- O. Danvy and A. Filinski. Abstracting control. In *LISP and Functional Programming*, pages 151–160, 1990.
- A. Filinski. Representing monads. In *POPL*, pages 446–457, 1994.
- T. Griffin. A formulae-as-types notion of control. In *POPL*, pages 47–58, 1990.
- Y. Kameyama and M. Hasegawa. A sound and complete axiomatization of delimited continuations. In *ICFP*, pages 177–188, 2003.
- E. Moggi. Computational  $\lambda$ -calculus and monads. In *Logic in Computer Science*, 1989.

## References II

- M. Parigot. Lambda-my-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR*, pages 190–201, 1992.
- A. Sabry. Note on axiomatizing the semantics of control operators. Technical Report CIS-TR-96-03, Department of Computer and Information Science, University of Oregon, 1996.
- A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4): 289–360, 1993.