

# CONTROLLING COPATTERNS

THERE AND BACK AGAIN

---

**Paul Downen**

University of Massachusetts Lowell

OlivierFest — Wednesday, October 15, 2025

# THE CONTEXT

---

# QUICK PRIMER ON COPATTERNS

IN SCHEME / RACKET

Defining the  $2 \times 2$  matrix

$$\text{quad} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

as the nested pair  $((1, 2), (3, 4))$  by copattern matching:

```
(define*
  [((quad 'fst) 'fst) = 1]
  [((quad 'fst) 'snd) = 2]
  [((quad 'snd) 'fst) = 3]
  [((quad 'snd) 'snd) = 4])
```

# MIXING COPATTERNS OF DIFFERENT DEPTHS

LIKE A “WILDCARD” FOR BIGGER CONTEXTS

Overriding the diagonals (upper left, lower right):  $\left( \text{diag } x \ y \begin{bmatrix} a & b \\ c & d \end{bmatrix} \right) = \begin{bmatrix} x & b \\ c & y \end{bmatrix}$

(**define\***

```
[(((diag x y z) 'fst) 'fst) = x]  
[(((diag x y z) 'snd) 'snd) = y]  
[  (diag x y z)              = z])
```

is equivalent to (using equational reasoning) expanding the remaining options

(**define\***

```
[(((diag x y z) 'fst) 'fst) = x]  
[(((diag x y z) 'snd) 'snd) = y]  
[(((diag x y z) 'fst) 'snd) = ((z 'fst) 'snd)]  
[(((diag x y z) 'snd) 'fst) = ((z 'snd) 'fst)])
```

# COMPOSABLE COPATTERNS

## COMBINING PROGRAMS ALONG DIFFERENT DIMENSIONS

- Programs defined by equational reasoning on their context (à la ML, Haskell)
- Composition of extensible fragments at run-time
  - **Vertical** — *either or* — compose alternative options, handling failure
  - **Horizontal** — *and then* — compose sequence of steps, parameters, matching, guards
  - **Circular** — *self* — recursion back on the entire composition itself
- Side benefit: supports infinite objects, some OO-style designs

# AN EQUATIONAL ARITHMETIC EVALUATOR

IS THIS ML, OR SCHEME?

```
; Expr = number | `(add ,Expr ,Expr) | `(mul ,Expr ,Expr)
```

```
; expr0 : Expr
```

```
(define expr0 '(add 1 (mul 2 3)))
```

```
; (arith `eval Expr) : number
```

```
(define-object
```

```
  [(arith 'eval n) (try-if (number? n))  
    = n]
```

```
  [(arith 'eval `(add ,l ,r))  
    = (+ (arith 'eval l) (arith 'eval r))]
```

```
  [(arith 'eval `(mul ,l ,r))  
    = (* (arith 'eval l) (arith 'eval r))])
```

```
(arith 'eval expr0) = 7
```

# EXTENDING EVALUATION WITH A NEW OPERATOR

THE WRONG WAY

```
; Expr = ... / `(neg ,Expr)

; expr1 : Expr
(define expr1 '(add 1 (neg (mul 2 3))))

(define-object
  [(arith-wrong 'eval '(neg ,e))
   = (- (arith-wrong 'eval e))]
  [(arith-wrong 'eval e) = (arith 'eval e)])

(arith-wrong 'eval expr1)
=
(+ 1 (arith 'eval '(neg (mul 2 3))))
=/=
```

# EXTENDING EVALUATION WITH A NEW OPERATOR

THE CORRECT WAY, USING VERTICAL COMPOSITION

```
; Expr = ... / `(neg ,Expr)
```

```
; expr1 : Expr
```

```
(define expr1 '(add 1 (neg (mul 2 3))))
```

Correct vertical composition:

```
; (arith-ext `eval Expr) : number
```

```
(define arith-ext
```

```
  (arith 'compose
```

```
    (object
```

```
      [(self 'eval '(neg ,e))
```

```
        = (- (self 'eval e))]))))
```

```
(arith-ext 'eval expr1)
```

```
=
```

```
(+ 1 (arith-ext 'eval '(neg (mul 2 3))))
```

```
=
```

```
-5
```



# UNDERSTANDING THE RESULT OF COMPOSITION

USING EQUATIONAL REASONING

Expanding the vertical composition:

(**define-object**

```
[(arith-ext 'eval n) (try-if (number? n))  
  = n]  
[(arith-ext 'eval '(add ,l ,r))  
  = (+ (arith-ext 'eval l) (arith-ext 'eval r))]  
[(arith-ext 'eval '(mul ,l ,r))  
  = (* (arith-ext 'eval l) (arith-ext 'eval r))]  
[(arith-ext 'eval '(neg ,e))  
  = (- (arith-ext 'eval e))]
```

# How Do We Extend Arithmetic to Algebra?

A MORE SERIOUS EXTENSION, NEEDS ANOTHER PARAMETER

```
; Expr = ... / symbol  
  
; expr2 : Expr  
(define expr2 '(add x (neg (mul 2 y))))  
  
; Env = ((symbol . number ) ...)  
  
; env-xy : Env  
(define env-xy '((x . 10) (y . 20)))  
  
(define-object alg ...?)
```

# EXTENDING THE EVALUATOR WITH AN ENVIRONMENT!

IS THIS FUNCTIONAL, OR OBJECT-ORIENTED?

```
; ((with-env Env) `env) : Env
(define (with-env dict)
  (object [(_ `env) = dict]))
```

```
; ((alg Env) `env) : Env
; ((alg Env) `eval Expr) : number
(define (alg dict)
  (arith-ext 'compose
    (with-env dict)
    (object
      [(self `eval x) (try-if (symbol? x))
        = (dict-ref (self `env) x)])))
```

```
((alg env-xy) `eval expr0) = 7
((alg env-xy) `eval expr1) = -5
((alg env-xy) `eval expr2) = -30
```

# THE PROBLEM

---

- Compositional copatterns implemented as Scheme / Racket macros
- For free: semantics as (selective) CPS
- This gives some rules for equational reasoning...

- Compositional copatterns implemented as Scheme / Racket macros
- For free: semantics as (selective) CPS
- This gives some rules for equational reasoning...
  - ...but not enough to calculate every answer
- Really need a standard operational semantics
  - Reason about copattern-matching programs directly (no translation)
  - State and prove type safety problems

# OPERATIONAL SEMANTICS OF COMPOSITIONAL COPATTERNS

WHAT TO DO, WHAT TO DO...?

- How the heck to deal with all the recursive generality and edge cases?!

# OPERATIONAL SEMANTICS OF COMPOSITIONAL COPATTERNS

WHAT TO DO, WHAT TO DO...?

- How the heck to deal with all the recursive generality and edge cases?!
- Oh wait! I remember! Olivier taught me exactly how to deal with this kind of thing!



# THE UNITY OF SEMANTIC ARTIFACTS

- Technique for mechanically deriving one style of semantics from another
- Using only off-the-shelf, semantics-preserving program transformations
- Correspondence between semantic artifacts for free!
- Many good intros: “A Walk in the Semantic Park”<sup>1</sup> is quite refreshing

---

<sup>1</sup>Olivier Danvy, Jacob Johannsen, and Ian Zerny, PEPM 2011.

# THE UNITY OF SEMANTIC ARTIFACTS

- Technique for mechanically deriving one style of semantics from another
- Using only off-the-shelf, semantics-preserving program transformations
- Correspondence between semantic artifacts for free!
- Many good intros: “A Walk in the Semantic Park”<sup>1</sup> is quite refreshing
- This was a key part of my second academic paper:

## Classical Call-by-Need Sequent Calculi: The Unity of Semantic Artifacts

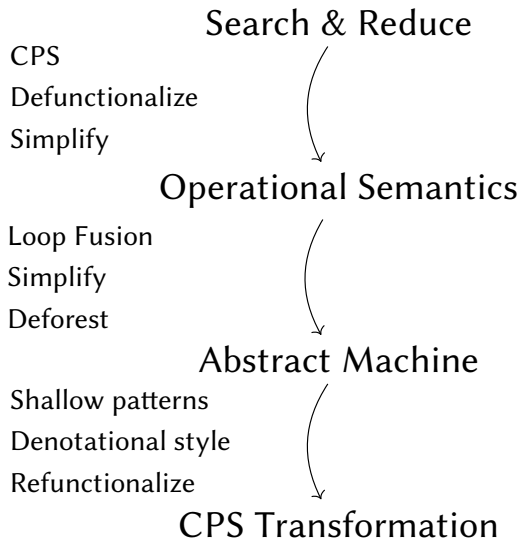
Zena M. Ariola<sup>1</sup>, Paul Downen<sup>1</sup>, Hugo Herbelin<sup>2</sup>, Keiko Nakata<sup>3</sup>,  
and Alexis Saurin<sup>2</sup>

**Abstract.** We systematically derive a classical call-by-need sequent calculus, which does not require an unbounded search for the standard re-dex, by using the unity of semantic artifacts proposed by Danvy *et al.* The calculus serves as an intermediate step toward the generation of an environment-based abstract machine. The resulting abstract machine is context-free, so that each step is parametric in all but one component. The context-free machine elegantly leads to an environment-based CPS

---

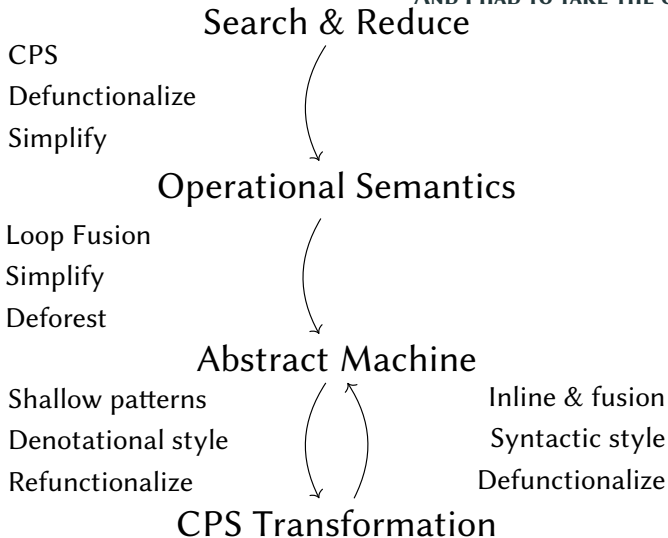
<sup>1</sup>Olivier Danvy, Jacob Johannsen, and Ian Zerny, PEPM 2011.

# TWO ROADS THROUGH THE SEMANTIC WOODS



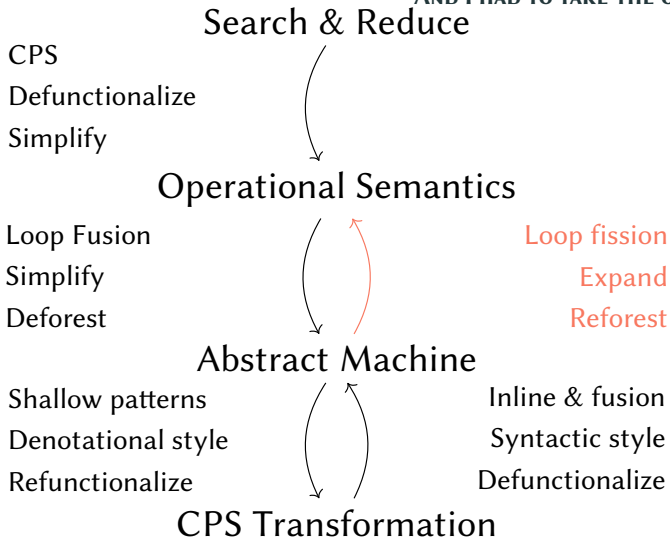
# TWO ROADS THROUGH THE SEMANTIC WOODS

AND I HAD TO TAKE THE ONE LESS TRAVELED BY



# TWO ROADS THROUGH THE SEMANTIC WOODS

AND I HAD TO TAKE THE ONE LESS TRAVELED BY



# THE SOLUTION

---

# A ROUND-TRIP IN THE SEMANTIC PARK

## TAKING THE EASY PATH TO RETRACE OUR STEPS

- “Reforestation” and loop “fission” are hard
- Why? They undo lossy transformations
  - Loop fusion destroys the inner loops
  - Deforestation destroys intermediate data structures
- It helps to see what was lost to restore it
- Trick: First take the easy “forward” path from a similar starting point to light the way for the trip back

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$\begin{aligned} (\beta) \quad C[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] &= M_j[\overrightarrow{N/x}] \\ &\left( \begin{array}{l} \text{if} \quad C = L_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right) \end{aligned}$$



# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid \textcolor{red}{M} \textcolor{red}{N} \mid M X \mid \lambda\{L \rightarrow M\ldots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$\begin{aligned} (\beta) \quad C[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] &= M_j[\overrightarrow{N/x}] \\ &\left( \begin{array}{l} \text{if} \quad C = L_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right) \end{aligned}$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid \textcolor{red}{M} \textcolor{red}{X} \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$\begin{aligned} (\beta) \quad C[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] &= M_j[\overrightarrow{N/x}] \\ &\left( \begin{array}{l} \text{if} \quad C = L_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right) \end{aligned}$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} \ni M, N &::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M\ldots\} \\ \text{Copat} \ni L &::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$\begin{aligned} (\beta) \quad C[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] &= M_j[\overrightarrow{N/x}] \\ &\left( \begin{array}{l} \text{if} \quad C = L_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right) \end{aligned}$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$\begin{aligned} (\beta) \quad & \text{C}[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] = M_j[\overrightarrow{N/x}] \\ & \left( \begin{array}{l} \text{if} \quad C = L_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right) \end{aligned}$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$(\beta) \quad \textcolor{red}{C}[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] = \textcolor{red}{M}_j[\overrightarrow{N/x}] \left( \begin{array}{l} \text{if} \quad \textcolor{green}{C} = \textcolor{green}{L}_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, C = L_i[\overrightarrow{N/x}] \end{array} \right)$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$(\beta) \quad \textcolor{red}{C}[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] = \textcolor{red}{M}_j[\overrightarrow{N/x}] \quad \left( \begin{array}{l} \text{if} \quad \textcolor{green}{C} = \textcolor{green}{L}_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, \textcolor{magenta}{C} = \textcolor{magenta}{L}_i[\overrightarrow{N/x}] \end{array} \right)$$

# A NAÏVE SEMANTICS FOR MONOLITHIC COPATTERNS

COPATTERNS = EQUATIONAL REASONING ON THE CONTEXT

$$\begin{aligned} \text{Term} &\ni M, N ::= x \mid M N \mid M X \mid \lambda\{L \rightarrow M \dots\} \\ \text{Copat} &\ni L ::= \varepsilon \mid x L \mid X L \end{aligned}$$

$$(\beta) \quad \textcolor{red}{C}[\lambda\{L_i \rightarrow M_i^{1 \leq i \leq n}\}] = \textcolor{red}{M}_j[\overrightarrow{N/x}] \quad \left( \begin{array}{l} \text{if} \quad \textcolor{green}{C} = \textcolor{green}{L}_j[\overrightarrow{N/x}] \\ \text{and } \forall i < j, \nexists \overrightarrow{N}, \textcolor{magenta}{C} = \textcolor{magenta}{L}_i[\overrightarrow{N/x}] \end{array} \right)$$

# WRITING AN ALGORITHM FOR SMALL STEPS

```
-- Syntax
data Term      i a -- `i` represents literal index, `a` represents variable
data Copattern i a
type Question  i a = Copattern i (Term i a) -- copattern-shaped contexts
```



# WRITING AN ALGORITHM FOR SMALL STEPS

*-- Syntax*

**data Term** i a -- *`i` represents literal index, `a` represents variable*

**data Copattern** i a

**type Question** i a = **Copattern** i (**Term** i a) -- *copattern-shaped contexts*

*-- Reduction*

**data Redex** i a = **Respond** [**Option** i a] | **FreeVar** a

**data Reduct** i a = **Reduced** (**Term** i a) | **Unhandled** | **Unknown** a

**data Followup** i a = **Next** (**Reduct** i a) (**Question** i a)  
| **More** (**Copattern** i a) (**Term** i a)  
| [**Option** i a] (**Question** i a)

**reduce** :: (**Eq** i, **Eq** a) => **Redex** i a -> **Question** i a -> **Followup** i a

# WRITING AN ALGORITHM FOR SMALL STEPS

*-- Syntax*

**data Term** i a -- *`i` represents literal index, `a` represents variable*

**data Copattern** i a

**type Question** i a = **Copattern** i (**Term** i a) -- *copattern-shaped contexts*

*-- Reduction*

**data Redex** i a = **Respond** [**Option** i a] | **FreeVar** a

**data Reduct** i a = **Reduced** (**Term** i a) | **Unhandled** | **Unknown** a

**data Followup** i a = **Next** (**Reduct** i a) (**Question** i a)  
| **More** (**Copattern** i a) (**Term** i a)  
| [**Option** i a] (**Question** i a)

**reduce** :: (**Eq** i, **Eq** a) => **Redex** i a -> **Question** i a -> **Followup** i a

*-- Search*

**data Found** i a = **Asked** (**Redex** i a) (**Question** i a)

**search** :: **Term** i a -> **Found** i a

A few moments later ...

# THE DERIVED CPS TRANSFORMATION

A JOURNEY OF SMALL STEPS TO THE LAND OF CONTINUATIONS

Applicative forms look like normal CBN CPS:

$$\begin{aligned}\llbracket x \rrbracket &= x \\ \llbracket M N \rrbracket &= \lambda k. \llbracket M \rrbracket (\llbracket N \rrbracket, k) \\ \llbracket M X \rrbracket &= \lambda k. \llbracket M \rrbracket (X k)\end{aligned}$$

$\lambda$ s begin copattern-matching various options against the continuation:

$$\begin{aligned}\llbracket \lambda \{ \varepsilon \} \rrbracket &= \lambda k. k \\ \llbracket \lambda \{ L \rightarrow M \mid L' \rightarrow M' \dots \} \rrbracket &= \lambda k. \llbracket L \rightarrow M \rrbracket k \llbracket \lambda \{ L' \rightarrow M' \dots \} \rrbracket k\end{aligned}$$

# THE DERIVED CPS TRANSFORMATION

## A JOURNEY OF SMALL STEPS TO THE LAND OF CONTINUATIONS

Copatterns match on the given continuation:

$$\begin{aligned}\llbracket \varepsilon \rightarrow N \rrbracket &= \lambda q. \lambda f. \llbracket N \rrbracket \\ \llbracket x L \rightarrow N \rrbracket &= \mathbf{rec} \, r = \lambda q. \lambda f. \lambda k. \\ &\quad \mathbf{case} \, k \, \mathbf{of} \, \begin{array}{ll} (x, k') & \rightarrow \llbracket L \rightarrow N \rrbracket \, q \, f \, k' \\ () & \rightarrow r \, q \, f \\ k & \rightarrow f \, q \end{array} \\ \llbracket X L \rightarrow N \rrbracket &= \mathbf{rec} \, r = \lambda q. \lambda f. \lambda k. \\ &\quad \mathbf{case} \, k \, \mathbf{of} \, \begin{array}{ll} (X \, k') & \rightarrow \llbracket L \rightarrow N \rrbracket \, q \, f \, k' \\ () & \rightarrow r \, q \, f \\ k & \rightarrow f \, q \end{array}\end{aligned}$$

# THE DERIVED CPS TRANSFORMATION

## A JOURNEY OF SMALL STEPS TO THE LAND OF CONTINUATIONS

Copatterns match on the given continuation:

$$\begin{aligned}\llbracket \varepsilon \rightarrow N \rrbracket &= \lambda q. \lambda f. \llbracket N \rrbracket \\ \llbracket x L \rightarrow N \rrbracket &= \text{rec } r = \lambda q. \lambda f. \lambda k. \\ &\quad \text{case } k \text{ of } (x, k') \rightarrow \llbracket L \rightarrow N \rrbracket \ q \ f \ k' \\ &\quad \quad \quad () \rightarrow r \ q \ f \\ &\quad \quad \quad k \rightarrow f \ q \\ \llbracket X L \rightarrow N \rrbracket &= \text{rec } r = \lambda q. \lambda f. \lambda k. \\ &\quad \text{case } k \text{ of } (X \ k') \rightarrow \llbracket L \rightarrow N \rrbracket \ q \ f \ k' \\ &\quad \quad \quad () \rightarrow r \ q \ f \\ &\quad \quad \quad k \rightarrow f \ q\end{aligned}$$

- If the question is too short, check again when given more continuation

# THE DERIVED CPS TRANSFORMATION

## A JOURNEY OF SMALL STEPS TO THE LAND OF CONTINUATIONS

Copatterns match on the given continuation:

$$\begin{aligned}\llbracket \varepsilon \rightarrow N \rrbracket &= \lambda q. \lambda f. \lambda k. \llbracket N \rrbracket k \\ \llbracket x L \rightarrow N \rrbracket &= \mathbf{rec} \, r = \lambda q. \lambda f. \lambda k. \\ &\quad \mathbf{case} \, k \, \mathbf{of} \, (x, k') \rightarrow \llbracket L \rightarrow N \rrbracket \, q \, f \, k' \\ &\quad \quad \quad () \rightarrow r \, q \, f \\ &\quad \quad \quad k \rightarrow f \, q \\ \llbracket X L \rightarrow N \rrbracket &= \mathbf{rec} \, r = \lambda q. \lambda f. \lambda k. \\ &\quad \mathbf{case} \, k \, \mathbf{of} \, (X \, k') \rightarrow \llbracket L \rightarrow N \rrbracket \, q \, f \, k' \\ &\quad \quad \quad () \rightarrow r \, q \, f \\ &\quad \quad \quad k \rightarrow f \, q\end{aligned}$$

- If the question is too short, **check again when given more continuation**
- On successful match, **keep checking the rest of the continuation**
- On a mismatch failure, **reset to original question and try next option**

# BRIDGING THE GAP BETWEEN MONOLITHIC & COMPOSITIONAL

## A SHORT REST AMONG THE LAMBDAΣ

Equational reasoning to “clean up” the CPS to resemble compositional copattern macros:

- Delimiting the context
  - Add an explicit “end of message” marker ( $M ! R$ ) at the end of the question
  - Avoids confusion between “not enough context” versus “question too short”
- Nesting copatterns
  - Regroup copatterns ( $L \rightarrow M$ ) to single steps that lean to the right ( $O$ )
  - Add an explicit “alternative” ( $O ? M$ ) for when matching fails
- Eliminate redundancy
  - Instead of passing failure+alternative+success continuations, fold failure into alternative
  - $\llbracket \lambda \{ L \rightarrow M \mid L' \rightarrow M' \dots \} \rrbracket = \lambda k. \llbracket L \rightarrow M \rrbracket \textcolor{violet}{k} \llbracket \lambda \{ L' \rightarrow M' \dots \} \rrbracket \textcolor{teal}{k}$
- Regain proper CPS through double-barrel continuations (like Shift+Reset)



# BRIDGING THE GAP BETWEEN MONOLITHIC & COMPOSITIONAL

## A SHORT REST AMONG THE LAMBDAΣ

Equational reasoning to “clean up” the CPS to resemble compositional copattern macros:

- Delimiting the context
  - Add an explicit “end of message” marker ( $M ! R$ ) at the end of the question
  - Avoids confusion between “not enough context” versus “question too short”
- Nesting copatterns
  - Regroup copatterns ( $L \rightarrow M$ ) to single steps that lean to the right ( $O$ )
  - Add an explicit “alternative” ( $O ? M$ ) for when matching fails
- Eliminate redundancy
  - Instead of passing failure+alternative+success continuations, fold failure into alternative
  - $\llbracket \lambda \{ L \rightarrow M \mid L' \rightarrow M' \dots \} \rrbracket = \lambda k. \llbracket L \rightarrow M \rrbracket \llbracket \lambda \{ L' \rightarrow M' \dots \} \rrbracket k$
- Regain proper CPS through double-barrel continuations (like Shift+Reset)

# A CALCULUS FOR COMPOSITIONAL COPATTERNS WITH CONTROL

$$\text{Response } \ni R ::= q \mid \varepsilon \mid M!R$$

$$\text{Term } \ni M, N ::= x \mid MN \mid MX \mid M. \mid \mathbf{raise} \mid O?M \mid !q \rightarrow R$$

$$\text{Option } \ni O ::= x \rightarrow O \mid X \rightarrow O \mid ?x \rightarrow M$$

Old monolithic syntax now just sugar (proved correct by CPS!) on smaller primitives:

$$\lambda\{O_1 \mid \cdots \mid O_n\} := O_1 ? (\cdots ? (O_n ? \mathbf{raise}))$$

$$\varepsilon \rightarrow M := ?_ \rightarrow M$$

$$(x L) \rightarrow O := x \rightarrow (L \rightarrow O)$$

$$(X L) \rightarrow O := X \rightarrow (L \rightarrow O)$$

# A CALCULUS FOR COMPOSITIONAL COPATTERNS WITH CONTROL

$$\text{Response } \ni R ::= q \mid \varepsilon \mid M!R$$
$$\text{Term } \ni M, N ::= x \mid MN \mid MX \mid M. \mid \mathbf{raise} \mid O?M \mid !q \rightarrow R$$
$$\text{Option } \ni O ::= x \rightarrow O \mid X \rightarrow O \mid ?x \rightarrow M$$

Old monolithic syntax now just sugar (proved correct by CPS!) on smaller primitives:

$$\lambda\{O_1 \mid \cdots \mid O_n\} := O_1 ? (\cdots ? (O_n ? \mathbf{raise}))$$
$$\varepsilon \rightarrow M := ?\_ \rightarrow M$$
$$(x L) \rightarrow O := x \rightarrow (L \rightarrow O)$$
$$(X L) \rightarrow O := X \rightarrow (L \rightarrow O)$$

But smaller primitives now give more functionality, such as **vertical composition**

$$\text{object } O := \lambda\{O \mid \text{self } Open \rightarrow \lambda\{x \rightarrow O?x\}\}$$
$$\text{compose} := \lambda o o' \rightarrow \text{object } \{?x \rightarrow o.Open(o'.Open x)\}$$
$$\text{compose object}\{O\} \text{ object}\{O'\} = \text{object}\{O \mid O'\}$$

A few moments later ...

# THE DERIVED OPERATIONAL SEMANTICS

## THE RETURN VOYAGE BACK TO DIRECT STYLE

$$\begin{aligned} & (?x \rightarrow N) ? M \mapsto N[M/x] \\ & ((x \rightarrow O) ? M) N \mapsto O[N/x] ? (M N) \\ & ((X \rightarrow O) ? M) X \mapsto O ? (M X) \\ & \quad (P ? M) X \mapsto M X \qquad \text{(otherwise)} \\ & \quad (P ? M) N \mapsto M N \qquad \text{(otherwise)} \\ \\ & E[!k \rightarrow R] ! \varepsilon \mapsto R[(E[\mathbf{raise}] ! \varepsilon)/k] \\ & M ! (E[\mathbf{raise}] ! \varepsilon) \mapsto E[M] ! \varepsilon \\ & \quad (P ? M) ! \varepsilon \mapsto M ! \varepsilon \end{aligned}$$

# THE DENOUEMENT

---

# THANK YOU, OLIVIER!

- We met when I was but a green Ph.D. student
- Your generosity has taught me much
- I'm happy the influence is still alive today!

If you want to play with these toys for yourself

Semantic derivations & examples



[https://github.com/  
pdownen/derive-copat](https://github.com/pdownen/derive-copat)

Copatterns for Racket & R<sup>6</sup>RS



[https://github.com/  
pdownen/CoScheme](https://github.com/pdownen/CoScheme)