Environment Analysis via ΔCFA

Matthew Might   Olin Shivers

Georgia Tech

POPL 2006

Control-flow analysis is not enough

Problem

- Closure = λ term + environment;
- *e.g.*, `(`λ `() x)` + [$\mathtt{x} \mapsto 3$]
- CFA: good with control (what λ invoked from which call sites);
- ... not so good with environments.

Control-flow analysis is not enough

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Fact: `(λ () x)` flows to `(h)`.

Question: Safe to inline?

Control-flow analysis is not enough

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x))))
  (f 0 (f 3 #f)))
```

Fact: `(λ () x)` flows to `(h)`.

Question: Safe to inline?

Answer: No.

Why: Only *one* variable `x` in program;
but *multiple dynamic bindings*.

$(\lambda\ ()\ x) + [x \mapsto 0]$

v.

$(\lambda\ ()\ x) + [x \mapsto 3]$

Control-flow analysis is not enough

Folding infinite set of binding contours
down to finite set causes merging.
Can lead to unsound conclusions.

Problem: $|x| = |y|$ does not imply $x = y$

Why it matters

We frequently use closures as general “carriers” of data:

- Create closure at point *a*.
- Ship to point *b* and invoke.

a & *b* have same static scope and *same dynamic bindings* $\Rightarrow$

- inline closure’s code at *b* (super-β inlining),
- communicate data via shared context.

Avoid heap allocating & fetching data.

Why it matters

We frequently use closures as general “carriers” of data:

- Create closure at point *a*.
- Ship to point *b* and invoke.

a & *b* have same static scope and *same dynamic bindings* $\Rightarrow$

- inline closure’s code at *b* (super-β inlining),
- communicate data via shared context.

Avoid heap allocating & fetching data.

Need to reason about environment relationships
between two control points.

Tool 1: Procedure strings

Classic model (Sharir & Pnueli, Harrison)

- Program trace at procedure level
- String of procedure activation/deactivation actions

Actions

control: call/return

Intuition: call extends environment; return restores environment.

Tool 1: Procedure strings

Classic model (Sharir & Pnueli, Harrison)

- Program trace at procedure level
- String of procedure activation/deactivation actions

Actions

control: call/return

Intuition: call extends environment; return restores environment.

`(fact 1)`

call `fact` / call `zero?` / return `zero?` / call `-` / return `-` /
call `fact` / call `zero?` / return `zero?` / return `fact` /
call `*` / return `*` / return `fact`

Note: Call/return items nest like parens.

Problems with procedure strings

- In functional languages, not all calls have matching returns. (*e.g.*, iteration)
- Procedure strings designed for "large-grain" procedures.
- What about other control/env operators? (loops, conditionals, coroutines, continuations, . . .)

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ
sequencing	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ
sequencing	call to λ
conditional	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ
sequencing	call to λ
conditional	call to λ
exception	call to λ

Tool 2: CPS

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ
sequencing	call to λ
conditional	call to λ
exception	call to λ
coroutine	call to λ
$\vdots$	$\vdots$

Now λ is fine-grained construct.

Adapt procedure-string models to CPS $\Rightarrow$
have universal analysis.

CPS & stacks

But wait! CPS is all calls, no returns!

Procedure strings won't nest properly:

call `a` / call `b` / call `c` / call `d` / . . .

CPS & stacks

But wait! CPS is all calls, no returns!

Procedure strings won't nest properly:
call `a` / call `b` / call `c` / call `d` / ...

Unless...

CPS & stacks

Solution

- Syntactically partition CPS language into "user" & "continuation" world.

  We still have calls & returns;
  have just decoupled them somewhat.
- Shift from call/return view
  to push/pop view.

  Calls & returns no longer nest,
  but pushes & pops *always* nest.

Example: recursive factorial

```
(λ (n)
  (letrec ((f (λ (m)
                (if0 m 1
                     (* m (f (- m 1)))))))
    (f n)))
```

Example: recursive factorial

```
(λ_t (n ktop)
  (letrec ((f (λ_f (m k)
                (%if0 m
                  (λ_1 () (k 1))
                  (λ_2 ()
                    (- m 1 (λ_3 (m2)
                             (f m2 (λ_4 (a)
                                     (* m a k)
                                     ))))))))))
    (f n ktop)))
```

Example: recursive factorial

```
(λt (n ktop)
  (letrec ((f (λf (m k)
                (%if0 m
                  (λ1 () (k 1))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (f m2 (λ4 (a)
                                     (* m a k)
                                     ))))))))))
    (f n ktop)))
```

But...

Blue ≠ call/push

Red ≠ return/pop

Putting it all together: frame strings

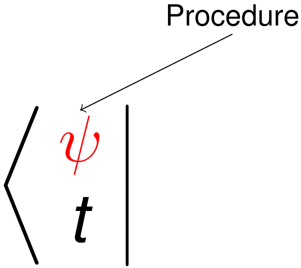
Frame strings, *F*

- Record push/pop sequences.
- Each character: push or pop.
- Calls push frames.
- Continuations restore stacks.

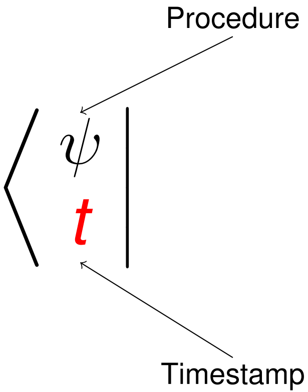
Anatomy of a push character

$$\left\langle \begin{matrix} \psi \\ \boldsymbol{t} \end{matrix} \right|$$

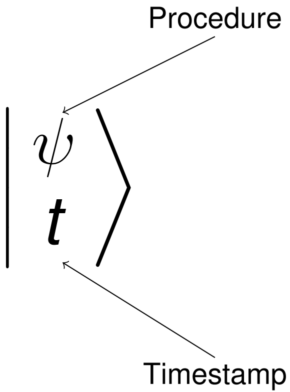
Anatomy of a push character



Anatomy of a push character



Anatomy of a pop character



Net & inverse operators

Net

- Written $\lfloor p \rfloor$.
- Cancels opposite neighbors.

Examples

- $\lfloor \langle\substack{\mathtt{a}\\ \mathtt{6}}| \langle\substack{\mathtt{b}\\ \mathtt{7}}| \substack{\mathtt{b}\\ \mathtt{7}}\rangle |\substack{\mathtt{a}\\ \mathtt{6}}\rangle \rfloor = \epsilon$
- $\lfloor |\substack{\mathtt{q}\\ \mathtt{38}}\rangle \langle\substack{\mathtt{q}\\ \mathtt{38}}| \rfloor = \epsilon$
- $\lfloor \langle\substack{\mathtt{r}\\ \mathtt{21}}| \substack{\mathtt{r}\\ \mathtt{21}}\rangle \langle\substack{\mathtt{a}\\ \mathtt{71}}| \rfloor = \langle\substack{\mathtt{a}\\ \mathtt{71}}|$

Net & inverse operators

Net

- Written $\lfloor p \rfloor$.
- Cancels opposite neighbors.

Examples

- $\lfloor \langle^{\mathtt{a}}_{6}| \langle^{\mathtt{b}}_{7}|^{\mathtt{b}}_{7}\rangle |^{\mathtt{a}}_{6}\rangle \rfloor = \epsilon$
- $\lfloor |^{\mathtt{q}}_{38}\rangle \langle^{\mathtt{q}}_{38}| \rfloor = \epsilon$
- $\lfloor \langle^{\mathtt{r}}_{21}|^{\mathtt{r}}_{21}\rangle \langle^{\mathtt{a}}_{71}| \rfloor = \langle^{\mathtt{a}}_{71}|$

Two views

Absolute $\lfloor p_t \rfloor$ is picture of stack at time t.

Relative $\lfloor p_t^{t'} \rfloor$ is summary of stack change.

Net & inverse operators

p^{-1} = reverse $\lfloor p \rfloor$ and swap "push" & "pop":

Example

$\left(\langle{a \atop 4}|\,\langle{b \atop 5}|{b \atop 5}\rangle\,\langle{c \atop 6}|\right)^{-1} = |{c \atop 6}\rangle\,|{a \atop 4}\rangle$

Frame strings mod $\lfloor\cdot\rfloor$ is group: $p + p^{-1} \equiv p^{-1} + p \equiv \epsilon$.
($+$ is concatenation)

The inverse operator

Use: restoring stack to previous state

Time	Frame string	Stack
t_1	p	$\lfloor p \rfloor$
t_2	$p + q$	$\lfloor p + q \rfloor$
t_3	$p + q + ???$	$\lfloor p \rfloor$

The inverse operator

Use: restoring stack to previous state

Time	Frame string	Stack
t_1	p	$\lfloor p \rfloor$
t_2	$p + q$	$\lfloor p + q \rfloor$
t_3	$p + q + q^{-1}$	$\lfloor p \rfloor$

This is what continuations do in CPS. . .
but expressed in terms of *change*.

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                      (f m2 a2 k)
                                      )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                      (f m2 a2 k)
                                      ))))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{t}_{1}\rvert$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                      (f m2 a2 k)
                                      ))))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
(f n 1 ktop)	tail call to λ_f	$\lvert^{t}_{1}\rangle\langle^{f}_{2}\rvert$	$\langle^{t}_{1}\rvert$ $\langle^{f}_{2}\rvert$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                      (f m2 a2 k)
                                      )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{t}_{1}\vert$
`(f n 1 ktop)`	tail call to λ_{f}	$\vert^{t}_{1}\rangle\langle^{f}_{2}\vert$	$\langle^{f}_{2}\vert$
`(%if0 m ...)`	call to `%if0`	$\langle^{\%if0}_{3}\vert$	$\langle^{f}_{2}\vert\langle^{\%if0}_{3}\vert$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                       (f m2 a2 k)
                                       )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{t}_{1}\|$
`(f n 1 ktop)`	tail call to $\lambda_{\mathtt{f}}$	$\|^{t}_{1}\rangle\langle^{f}_{2}\|$	$\langle^{f}_{2}\|$
`(%if0 m ...)`	call to `%if0`	$\langle^{\%if0}_{3}\|$	$\langle^{f}_{2}\|\langle^{\%if0}_{3}\|$
`%if0` *internal*	return to $\underline{\lambda}_2$	$\|^{\%if0}_{3}\rangle\langle^{2}_{4}\|$	$\langle^{f}_{2}\|\langle^{2}_{4}\|$

Iterative factorial example

```
(λ_t (n ktop)
  (letrec ((f (λ_f (m a k)
                (%if0 m
                  (λ_1 () (k a))
                  (λ_2 ()
                    (- m 1 (λ_3 (m2)
                             (* m a (λ_4 (a2)
                                      (f m2 a2 k)
                                      )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{t}_{1}\vert$
`(f n 1 ktop)`	tail call to λ_f	$\vert^{t}_{1}\rangle\langle^{f}_{2}\vert$	$\langle^{f}_{2}\vert$
`(%if0 m ...)`	call to `%if0`	$\langle^{\%if0}_{3}\vert$	$\langle^{f}_{2}\vert\langle^{\%if0}_{3}\vert$
`%if0` *internal*	return to $\underline{\lambda}_2$	$\vert^{\%if0}_{3}\rangle\langle^{2}_{4}\vert$	$\langle^{f}_{2}\vert\langle^{2}_{4}\vert$
`(- m 1 ...)`	call to `-`	$\langle^{-}_{5}\vert$	$\langle^{f}_{2}\vert\langle^{2}_{4}\vert\langle^{-}_{5}\vert$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                       (f m2 a2 k)
                                       )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{t}_{1}\vert$
`(f n 1 ktop)`	tail call to λ_{f}	$\vert^{t}_{1}\rangle\langle^{f}_{2}\vert$	$\langle^{f}_{2}\vert$
`(%if0 m ...)`	call to `%if0`	$\langle^{\%if0}_{3}\vert$	$\langle^{f}_{2}\vert\langle^{\%if0}_{3}\vert$
`%if0` *internal*	return to $\underline{\lambda}_2$	$\vert^{\%if0}_{3}\rangle\langle^{2}_{4}\vert$	$\langle^{f}_{2}\vert\langle^{2}_{4}\vert$
`(- m 1 ...)`	call to `-`	$\langle^{-}_{5}\vert$	$\langle^{f}_{2}\vert\langle^{2}_{4}\vert\langle^{-}_{5}\vert$
`-` *internal*	return to $\underline{\lambda}_3$	$\vert^{-}_{5}\rangle\langle^{3}_{6}\vert$	$\langle^{f}_{2}\vert\langle^{2}_{4}\vert\langle^{3}_{6}\vert$

Iterative factorial example

```
(λ_t (n ktop)
  (letrec ((f (λ_f (m a k)
                (%if0 m
                  (λ_1 () (k a))
                  (λ_2 ()
                    (- m 1 (λ_3 (m2)
                             (* m a (λ_4 (a2)
                                      (f m2 a2 k)
                                      )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle^{\mathtt{t}}_{1}\|$
`(f n 1 ktop)`	tail call to $\lambda_{\mathtt{f}}$	$\|^{\mathtt{t}}_{1}\rangle\langle^{\mathtt{f}}_{2}\|$	$\langle^{\mathtt{f}}_{2}\|$
`(%if0 m ...)`	call to `%if0`	$\langle^{\mathtt{\%if0}}_{3}\|$	$\langle^{\mathtt{f}}_{2}\|\langle^{\mathtt{\%if0}}_{3}\|$
`%if0` *internal*	return to $\underline{\lambda}_2$	$\|^{\mathtt{\%if0}}_{3}\rangle\langle^{2}_{4}\|$	$\langle^{\mathtt{f}}_{2}\|\langle^{2}_{4}\|$
`(- m 1 ...)`	call to -	$\langle^{-}_{5}\|$	$\langle^{\mathtt{f}}_{2}\|\langle^{2}_{4}\|\langle^{-}_{5}\|$
- *internal*	return to $\underline{\lambda}_3$	$\|^{-}_{5}\rangle\langle^{3}_{6}\|$	$\langle^{\mathtt{f}}_{2}\|\langle^{2}_{4}\|\langle^{3}_{6}\|$
`(* m a ...)`	call to *	$\langle^{*}_{7}\|$	$\langle^{\mathtt{f}}_{2}\|\langle^{2}_{4}\|\langle^{3}_{6}\|\langle^{*}_{7}\|$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
                (%if0 m
                  (λ1 () (k a))
                  (λ2 ()
                    (- m 1 (λ3 (m2)
                             (* m a (λ4 (a2)
                                      (f m2 a2 k)
                                      )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle{}^{t}_{1}\rvert$
`(f n 1 ktop)`	tail call to λ_f	$\lvert{}^{t}_{1}\rangle\langle{}^{f}_{2}\rvert$	$\langle{}^{f}_{2}\rvert$
`(%if0 m ...)`	call to `%if0`	$\langle{}^{\%if0}_{3}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{\%if0}_{3}\rvert$
`%if0` *internal*	return to $\underline{\lambda}_2$	$\lvert{}^{\%if0}_{3}\rangle\langle{}^{2}_{4}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{2}_{4}\rvert$
`(- m 1 ...)`	call to `-`	$\langle{}^{-}_{5}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{2}_{4}\rvert\langle{}^{-}_{5}\rvert$
`-` *internal*	return to $\underline{\lambda}_3$	$\lvert{}^{-}_{5}\rangle\langle{}^{3}_{6}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{2}_{4}\rvert\langle{}^{3}_{6}\rvert$
`(* m a ...)`	call to `*`	$\langle{}^{*}_{7}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{2}_{4}\rvert\langle{}^{3}_{6}\rvert\langle{}^{*}_{7}\rvert$
`*` *internal*	return to $\underline{\lambda}_4$	$\lvert{}^{*}_{7}\rangle\langle{}^{4}_{8}\rvert$	$\langle{}^{f}_{2}\rvert\langle{}^{2}_{4}\rvert\langle{}^{3}_{6}\rvert\langle{}^{4}_{8}\rvert$

Iterative factorial example

```
(λt (n ktop)
  (letrec ((f (λf (m a k)
               (%if0 m
                 (λ1 () (k a))
                 (λ2 ()
                   (- m 1 (λ3 (m2)
                            (* m a (λ4 (a2)
                                     (f m2 a2 k)
                                     )))))))))
    (f n 1 ktop)))
```

Call site	Description	Stack Δ	Stack
			$\langle{}^{\mathtt{t}}_{1}|$
`(f n 1 ktop)`	tail call to $\lambda_{\mathtt{f}}$	$|{}^{\mathtt{t}}_{1}\rangle\langle{}^{\mathtt{f}}_{2}|$	$\langle{}^{\mathtt{f}}_{2}|$
`(%if0 m ...)`	call to `%if0`	$\langle{}^{\mathtt{\%if0}}_{3}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{\mathtt{\%if0}}_{3}|$
`%if0` *internal*	return to $\underline{\lambda}_2$	$|{}^{\mathtt{\%if0}}_{3}\rangle\langle{}^{2}_{4}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{2}_{4}|$
`(- m 1 ...)`	call to `-`	$\langle{}^{-}_{5}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{2}_{4}|\langle{}^{-}_{5}|$
`-` *internal*	return to $\underline{\lambda}_3$	$|{}^{-}_{5}\rangle\langle{}^{3}_{6}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{2}_{4}|\langle{}^{3}_{6}|$
`(* m a ...)`	call to `*`	$\langle{}^{*}_{7}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{2}_{4}|\langle{}^{3}_{6}|\langle{}^{*}_{7}|$
`*` *internal*	return to $\underline{\lambda}_4$	$|{}^{*}_{7}\rangle\langle{}^{4}_{8}|$	$\langle{}^{\mathtt{f}}_{2}|\langle{}^{2}_{4}|\langle{}^{3}_{6}|\langle{}^{4}_{8}|$
`(f m2 a2 k)`	tail call to $\lambda_{\mathtt{f}}$	$|{}^{4}_{8}\rangle|{}^{3}_{6}\rangle|{}^{2}_{4}\rangle|{}^{\mathtt{f}}_{2}\rangle\langle{}^{\mathtt{f}}_{9}|$	$\langle{}^{\mathtt{f}}_{9}|$

Adding frame strings to concrete CPS semantics

Key steps

- Give states time stamps.
- Give states frame-string log, $\delta : \mathit{Time} \rightharpoonup F$.
  Log is "relative" definition.
  (Just what we need!)
- Give values "birthdates": creation time stamps.

Example

If δ_{13} is the log from time 13, then $\delta_{13}(7)$ is the frame-string change between time 7 and time 13.

To invoke continuation with birthday t_b

Perform $\delta(t_b)^{-1}$ on stack.
(That is, add $\delta(t_b)^{-1}$ to frame string.)

Interval notation for frame-string change

$$[t, t'] = \delta_{t'}(t)$$

That is, $[t, t']$ is the frame-string change between time t and t'.

A taste of environment theory

Theorem (Pinching Lemma)

No stack change between two times iff the times the same.

$$\lfloor [t_1, t_2] \rfloor = \epsilon \iff t_1 = t_2.$$

Theorem

Environments separated by continuation frame actions differ by the continuations' bindings.

$$\lfloor [t_0, t_2] + [t_1, t_2]^{-1} \rfloor = |^{\gamma_1}_{i_1}\rangle \cdots |^{\gamma_n}_{i_n}\rangle \langle^{\gamma'_1}_{t_1}| \cdots \langle^{\gamma'_n}_{t_m}| \Rightarrow \beta_{t_1}|\overline{B(\vec{\gamma'})} = \beta_{t_0}|\overline{B(\vec{\gamma})}.$$

(Notes: β's represent environments; inferring t_0/t_1 environment relationship from log at time t_2.)

Building ΔCFA

ΔCFA

- Straightforward abstract interpretation.
- Extends Harrison's abstract procedure strings.

Abstract frame strings

- Map from procedure to net change in procedure.
- Net change described by finite set of regular expressions.

$$\widehat{F} = \mathrm{ProcedureLabels} \rightarrow \mathcal{P}(\Delta)$$

$$\Delta = \{\epsilon, \langle{:}|, |{:}\rangle, \langle{:}|\langle{:}|^{+}, |{:}\rangle|{:}\rangle^{+}, |{:}\rangle^{+}\langle{:}|^{+}\}$$

ΔCFA: Eval

$$\frac{}{\left(\llbracket(f\ e^*\ q^*)_\kappa\rrbracket, \beta, ve, \quad t\right) \Rightarrow (proc, \mathbf{d}, \mathbf{c}, ve, \quad t)}$$

$$\text{where} \begin{cases} proc = \mathcal{A}\,\beta\,ve\,t\,f \\ d_i = \mathcal{A}\,\beta\,ve\,t\,e_i \\ c_j = \mathcal{A}\,\beta\,ve\,t\,q_j \\ \\ \\ \end{cases}$$

ΔCFA: Eval

$$\frac{}{\left(\llbracket (f\ e^*\ q^*)_\kappa \rrbracket, \beta, ve, \delta, t\right) \Rightarrow (proc, \mathbf{d}, \mathbf{c}, ve, \delta', t)}$$

$$\text{where} \begin{cases} proc = \mathcal{A}\,\beta\,ve\,t\,f \\ d_i = \mathcal{A}\,\beta\,ve\,t\,e_i \\ c_j = \mathcal{A}\,\beta\,ve\,t\,q_j \\ \nabla\varsigma = \begin{cases} (age_\delta\ proc)^{-1} & f \in EXPC \\ (youngest_\delta\ \mathbf{c})^{-1} & \text{otherwise} \end{cases} \\ \delta' = \delta + (\lambda t.\nabla\varsigma) \end{cases}$$

ΔCFA: Eval

$$\frac{}{(\llbracket (f\ e^*\ q^*)_\kappa \rrbracket, \hat{\beta}, \widehat{ve}, \hat{\delta}, \hat{t}) \leadsto (\widehat{proc}, \hat{\mathbf{d}}, \hat{\mathbf{c}}, \widehat{ve}, \hat{\delta}', \hat{t})}$$

$$\text{where } \begin{cases} \widehat{proc} \in \hat{\mathcal{A}}\ \hat{\beta}\ \widehat{ve}\ \hat{t}\ f \\ \hat{d}_i = \hat{\mathcal{A}}\ \hat{\beta}\ \widehat{ve}\ \hat{t}\ e_i \\ \hat{c}_i = \hat{\mathcal{A}}\ \hat{\beta}\ \widehat{ve}\ \hat{t}\ q_i \\ \Delta\hat{p} = \begin{cases} \left(\widehat{age}_{\hat{\delta}}\, \{\widehat{proc}\}\right)^{-1} & f \in EXPC \\ \left(\widehat{youngest}_{\hat{\delta}}\, \hat{\mathbf{c}}\right)^{-1} & \text{otherwise} \end{cases} \\ \hat{\delta}' = \hat{\delta} \oplus (\lambda \hat{t}.\Delta\hat{p}) \end{cases}$$

ΔCFA: Apply

$$\frac{length(\mathbf{d}) = length(\mathbf{u}) \qquad length(\mathbf{c}) = length(\mathbf{k})}{((\llbracket(\lambda_\psi \ (u^* \ k^*) \ call)\rrbracket, \beta, t_b), \mathbf{d}, \mathbf{c}, ve, \quad t) \Rightarrow (call, \beta', ve', \quad t')}$$

$$\text{where} \begin{cases} t' = tick(t) \\ \beta' = \beta[u_i \mapsto t', k_j \mapsto t'] \\ ve' = ve[(u_i, t') \mapsto d_i, (k_j, t') \mapsto c_j] \\ \\ \end{cases}$$

ΔCFA: Apply

$$\frac{\mathit{length}(\mathbf{d}) = \mathit{length}(\mathbf{u}) \qquad \mathit{length}(\mathbf{c}) = \mathit{length}(\mathbf{k})}{((\llbracket (\lambda_{\psi}\ (u^*\ k^*)\ \mathit{call}) \rrbracket, \beta, t_b), \mathbf{d}, \mathbf{c}, \mathit{ve}, \delta, t) \Rightarrow (\mathit{call}, \beta', \mathit{ve}', \delta', t')}$$

$$\text{where} \begin{cases} t' = \mathit{tick}(t) \\ \beta' = \beta[u_i \mapsto t', k_j \mapsto t'] \\ \mathit{ve}' = \mathit{ve}[(u_i, t') \mapsto d_i, (k_j, t') \mapsto c_j] \\ \nabla\varsigma = \langle^{\psi}_{t'}| \\ \delta' = (\delta + (\lambda t.\nabla\varsigma))[t' \mapsto \epsilon] \end{cases}$$

ΔCFA: Apply

$$\frac{length(\widehat{\mathbf{d}}) = length(\mathbf{u}) \qquad length(\widehat{\mathbf{c}}) = length(\mathbf{k})}{(([\![(\lambda_{\psi}\ (u^*\ k^*)\ call)]\!], \widehat{\beta}, \widehat{t_b}), \widehat{\mathbf{d}}, \widehat{\mathbf{c}}, \widehat{ve}, \widehat{\delta}, \widehat{t}) \rightsquigarrow (call, \widehat{\beta}', \widehat{ve}', \widehat{\delta}', \widehat{t}')}$$

$$\text{where} \begin{cases} \widehat{t}' = \widehat{tick}(\widehat{t}) \\ \widehat{\beta}' = \widehat{\beta}[u_i \mapsto \widehat{t}', k_j \mapsto \widehat{t}'] \\ \widehat{ve}' = \widehat{ve} \sqcup \left[(u_i, \widehat{t}') \mapsto \widehat{d}_i, (k_j, \widehat{t}') \mapsto \widehat{c}_j\right] \\ \Delta\widehat{p} = |\langle^{\psi}_{\widehat{t}'}|| \\ \widehat{\delta}' = \left(\widehat{\delta} \oplus (\lambda\widehat{t}.\Delta\widehat{p})\right) \sqcup \left[\widehat{t}' \mapsto |\epsilon|\right] \end{cases}$$

Correctness of ΔCFA

Theorem

ΔCFA simulates the concrete semantics.

$$\begin{array}{ccccc}
\varsigma & \xrightarrow{|\cdot|} & |\varsigma| & \xrightarrow{\sqsubseteq} & \hat{\varsigma} \\
\Rightarrow \big\downarrow & & & & \big\downarrow \leadsto \\
\varsigma' & \xrightarrow{|\cdot|} & |\varsigma'| & \xrightarrow{\sqsubseteq} & \hat{\varsigma}'
\end{array}$$

Concrete super-β inlining condition

When is it safe to inline λ term ψ' at call site κ'?

- All calls at κ' are to ψ'.
- Environment in closure $\equiv$ environment at κ'.

Concrete super-β inlining condition

When is it safe to inline λ term ψ' at call site κ'?

- All calls at κ' are to ψ'.
- Environment in closure $\equiv$ environment at κ'.

$$
\begin{aligned}
&\mathit{Inlinable}((\kappa', \psi'), \mathit{pr}) \iff \\
&\qquad \forall(\llbracket (f\ e^*\ q^*)_\kappa \rrbracket, \beta, \mathit{ve}, \delta, t) \in \mathcal{V}(\mathit{pr}) : \\
&\qquad\quad \text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \beta_b, t_b) = \mathcal{A}\,\beta\,\mathit{ve}\,t\,f \\
&\qquad\quad \text{then } \begin{cases} \psi = \psi' \\ \beta_b|\mathit{free}(L_{pr}(\psi')) = \beta|\mathit{free}(L_{pr}(\psi')) \end{cases}
\end{aligned}
$$

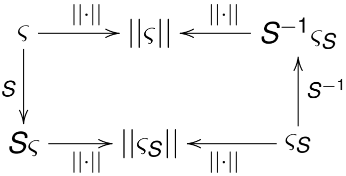
Correctness of super-β inlining

Theorem

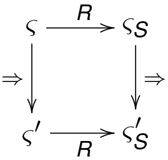
Inlining under Super-β condition does not change meaning.

Sketch of Proof.

Definition of R



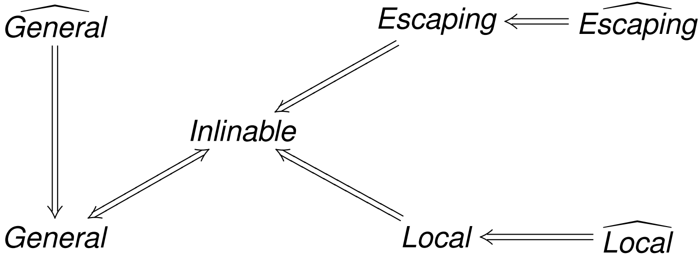
Bisimulation Relation



commutes.

□

Some ΔCFA super-β conditions



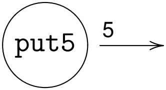
Implementation

- 3500 lines of Haskell.
- Direct-style front end for small Scheme.
- Choice of stack behavior models.
- Super-β inlining.
- β/η-reduction.
- Useless-variable elimination.
- Dead-code elimination.
- Sparse conditional constant propagation.
- Optimizes/fuses loops and co-routines.

A quick example: transducer/coroutine fusion

The put5 transducer

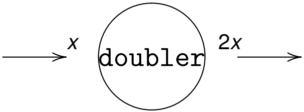
```
(letrec ((put5 (λ (chan)
                 (let ((chan (put 5 chan)))
                   (put5 chan)))))
  put5)
```



A quick example: transducer/coroutine fusion

The doubler transducer

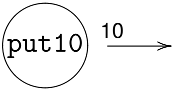
```
(letrec ((doubler (λ (uchan dchan)
                    (let* (((x uchan) (get uchan))
                           (dchan     (put (* 2 x) dchan)))
                      (doubler uchan dchan)))))
  doubler)
```



A quick example: transducer fusion

ΔCFA can fuse composed transducers into a single loop:

```
(compose/pull put5 doubler)
```



Still to come

- "Gradient" filtering.
- Contour GC.
- More experience with implementation.
- Context-free grammar or PDA abstractions for F?

Questions, Comments, Suggestions?

Question

What do you mean by “beyond the reach of β reduction?”

Answer

Certain loop-based optimizations are not possible with β reduction alone.

Example

```
(letrec ((lp1 (λ (i x)
               (if-zero i x
                (letrec ((lp2 (λ (j f y)
                               (if-zero j
                                (lp1 (- i 1) y)
                                (lp2 (- j 1)
                                     f
                                     [f y])))))
                 (lp2 10 [λ (n) (+ n i)] x))))))
  (lp1 10 0))
```

Question

What did you mean by frame strings “form a group”?

Answer

- Elements of group: Equivalence classes under net.
- Canonical member: The shortest.
- Identity element: $\{p : \lfloor p \rfloor = \epsilon\}$.
- + operator: Concatenate the cartesian product.
- Inverse: Invert every member of the class.

Concrete super-β I

$$
\begin{array}{l}
\textit{Local-Inlinable}((\kappa', \psi'), pr) \iff \\
\qquad \forall([\![(f\ e^*\ q^*)_\kappa]\!], \beta, ve, \delta, t) \in \mathcal{V}(pr) : \\
\qquad\qquad \text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \beta_b, t_b) = \mathcal{A}\,\beta\, ve\, t\, f \\
\qquad\qquad \text{then } \begin{cases} \psi = \psi' \\ \exists \vec{\gamma} : \begin{cases} \lfloor [t_b, t] \rfloor \succ^{\vec{\gamma}} \epsilon \\ \textit{free}(L_{pr}(\psi')) \subseteq \overline{B(\vec{\gamma})}. \end{cases} \end{cases}
\end{array}
$$

Abstract super-β I

$$\widehat{\mathit{Local\text{-}Inlinable}}((\kappa', \psi'), pr) \iff$$

$$\forall([\![(f\ e^*\ q^*)_\kappa]\!], \widehat{\beta}, \widehat{ve}, \widehat{\delta}, \widehat{t}) \in \widehat{\mathcal{V}}(pr):$$

$$\text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \widehat{\beta}_b, \widehat{t}_b) = \widehat{\mathcal{A}}\,\widehat{\beta}\,\widehat{ve}\,\widehat{t}\,f$$

$$\text{then } \begin{cases} \psi = \psi' \\ \exists \vec{\gamma} : \begin{cases} \widehat{\delta}(\widehat{t}_b) \succsim^{\vec{\gamma}} |\epsilon| \\ \mathit{free}(L_{pr}(\psi')) \subseteq \overline{B(\vec{\gamma})}. \end{cases} \end{cases}$$

Concrete super-β II

$$
\begin{array}{l}
\textit{Escaping-Inlinable}((\kappa', \psi'), pr) \iff \\
\quad \forall (\llbracket (f\ e^*\ q^*)_\kappa \rrbracket, \beta, ve, \delta, t) \in \mathcal{V}(pr) : \\
\quad\quad \text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \beta_b, t_b) = \mathcal{A}\,\beta\,ve\,t\,f \\
\quad\quad \text{then } \begin{cases} \psi = \psi' \\ \forall v \in free(L_{pr}(\psi)) : \exists \vec{\gamma} : \begin{cases} \lfloor [\beta(v), t] \rfloor \succ^{\vec{\gamma}} \lfloor [t_b, t] \rfloor \\ v \notin B(\vec{\gamma}). \end{cases} \end{cases}
\end{array}
$$

Abstract super-β II

$$\widehat{\textit{Escaping-Inlinable}}((\kappa', \psi'), pr) \iff$$

$$\forall (\llbracket (f\ e^*\ q^*)_\kappa \rrbracket, \widehat{\beta}, \widehat{ve}, \widehat{\delta}, \widehat{t}) \in \widehat{\mathcal{V}}(pr):$$

$$\text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \widehat{\beta}_b, \widehat{t}_b) = \widehat{\mathcal{A}}\,\widehat{\beta}\,\widehat{ve}\,\widehat{t}\,f$$

$$\text{then } \begin{cases} \psi = \psi' \\ \forall v \in \mathit{free}(L_{pr}(\psi)) : \exists \vec{\gamma} : \begin{cases} \widehat{\delta}(\widehat{\beta}(v)) \precsim^{\vec{\gamma}} \widehat{\delta}(\widehat{t}_b) \\ v \notin B(\vec{\gamma}). \end{cases} \end{cases}$$

Concrete super-β III

$$
\begin{aligned}
&\mathit{General\text{-}Inlinable}((\kappa', \psi'), pr) \iff \\
&\qquad \forall([\![(f\ e^*\ q^*)_\kappa]\!], \beta, ve, \delta, t) \in \mathcal{V}(pr) : \\
&\qquad\qquad \text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \beta_b, t_b) = \mathcal{A}\,\beta\,ve\,t\,f \\
&\qquad\qquad \text{then } \begin{cases} \psi = \psi' \\ \forall v \in free(L_{pr}(\psi)) : \lfloor[\beta(v), t]\rfloor = \lfloor[\beta_b(v), t]\rfloor. \end{cases}
\end{aligned}
$$

Abstract super-β III

$$\widehat{\textit{General-Inlinable}}((\kappa', \psi'), pr) \iff$$

$$\forall([\![(f\ e^*\ q^*)_\kappa]\!], \widehat{\beta}, \widehat{ve}, \widehat{\delta}, \widehat{t}) \in \widehat{\mathcal{V}}(pr) :$$

$$\text{if } \kappa = \kappa' \text{ and } (L_{pr}(\psi), \widehat{\beta}_b, \widehat{t}_b) = \widehat{\mathcal{A}}\,\widehat{\beta}\,\widehat{ve}\,\widehat{t}\,f$$

$$\text{then } \begin{cases} \psi = \psi' \\ \forall v \in \mathit{free}(L_{pr}(\psi)) : \widehat{\delta}(\widehat{\beta}(v)) \succsim^{\emptyset} \widehat{\delta}(\widehat{\beta}_b(v)). \end{cases}$$