Ordering Multiple Continuations on the Stack

Dimitrios Vardoulakis    Olin Shivers

Northeastern University
CPS in practice

CPS widely used in functional-language compilation.

Multiple continuations (conditionals, exceptions, etc).

Use a stack to manage them.
Contributions

- Syntactic restriction on multi-continuation CPS for better reasoning about stack.
- Static analysis for efficient multi-continuation CPS.
Overview

- Background:
  - Continuation-passing style (CPS)
  - Multi-continuation CPS
  - CPS with a runtime stack
- Restricted CPS (RCPS)
- Continuation-age analysis
- Evaluation
Continuation-passing style (CPS)

Characteristics

- Each function takes a continuation argument, “returns” by calling it.
- All intermediate computations are named.
- Continuations reified as lambdas.
Continuation-passing style (CPS)

Characteristics

▶ Each function takes a continuation argument, “returns” by calling it.
▶ All intermediate computations are named.
▶ Continuations reified as lambdas.

Example

(define (discr a b c)
  (- (* b b) (* 4 a c)))
Continuation-passing style (CPS)

Characteristics

▶ Each function takes a continuation argument, “returns” by calling it.
▶ All intermediate computations are named.
▶ Continuations reified as lambdas.

Example

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]

\[
\text{(define (discr a b c k)} \quad \text{CPS} \quad \text{(define (discr a b c k)} \\
\text{(- (* b b) (* 4 a c)))}
\]
Partitioned CPS [Steele 78, Rabbit]

\[
\text{(define (discr a b c k)}
\text{(\%* b b)}
\text{(\lambda (p1)}
\text{(\%* 4 a c)}
\text{(\lambda (p2)}
\text{(\%- p1 p2)}
\text{(\lambda (d) (k d))))))
\]

- Variables, lambdas and calls split into disjoint sets, “user” and “continuation”.
- Calls classified depending on operator.
Multi-continuation CPS

;; Add all positive numbers in the list
(define (add-pos l)
  (if (null? l)
      0
      (let ((fst (car l))
             (rest (cdr l)))
        (if (< 0 fst)
            (+ fst (add-pos rest))
            (add-pos rest))))
Multi-continuation CPS: Conditionals

(define (add-pos l k)
    ...)
(%if pos-fst
    (λ() (add-pos rest
        (λ(res) (%+ fst res k)))))
(λ() (add-pos rest k))))
Multi-continuation CPS: Exception handlers

(define (add-pos l k-ret k-exn)
  ...
  (λ(fst)
   (%number? fst
    (λ(num-fst)
     (%if num-fst
      (λ() ...)            
      (λ() (k-exn "Not a list of numbers.")))))))
Compile CPS without stack [Steele 78, Rabbit]

Argument evaluation pushes stack, function calls are jumps.

In CPS, every call is a tail call.

All closures in heap.

GC pressure.
Tail calls from direct style, 
continuation argument is a variable.

(define (add-pos l k)
  ... 
  (%if pos-fst
    (λ() (add-pos rest (λ(res) (%+ fst res k))))
    (λ() (add-pos rest k))))
Escaping continuations

\[(\lambda_1(f \ k) \ (k \ (\lambda_2(g \ k2) \ (g \ 42 \ k))))\]
Escaping continuations

\((\lambda_1(f \ k) \ (k \ (\lambda_2(g \ k2) \ (g \ 42 \ k))))\)

No capturing of continuation variables by user closures [Sabry-Felleisen 92], [Danvy-Lawall 92].
Restricted CPS (RCPS)

- A user lambda doesn’t contain free continuation variables,
- Or it’s $\alpha$-equivalent to $(\lambda(f \ cc)(f (\lambda(x \ k)(cc \ x)) \ cc))$
A user lambda doesn’t contain free continuation variables,
Or it’s $\alpha$-equivalent to $(\lambda(f \ cc)(f (\lambda(x \ k)(cc \ x)) \ cc))$

For example,

$$(\lambda_1(u \ k1 \ k2)(u (\lambda_2(k3)(k3 \ u)))
\k1
(\lambda_3(v)(k2 \ v))))$$
What does RCPS buy us?

Continuations escape in a controlled way.

Theorem: Continuations in argument position are stackable.
What does RCPS buy us?

Continuations escape in a controlled way.

Theorem: Continuations in argument position are stackable.
Proof?
The lifetime of a continuation argument

Doesn’t escape:

$$(((\lambda(u \ k) (k \ u))
  \ "foo") \ ✔
  clam)$$
The lifetime of a continuation argument

Operator, escapes:

\[=((\lambda(u\ cc)\ (f\ (\lambda(x\ k)\ (cc\ x))\ cc))\ (\lambda(v\ k)\ (k\ v))\ clamp)\]
The lifetime of a continuation argument

Argument, escapes:

\[((\lambda (k) \ (k \ (\lambda (u \ k2) \ (u \ k)))) \ \textit{clam}) \times\]
Extending the Orbit stack policy

Tail calls with multiple continuations:
(f e1 e2 k1 k2 k3)
Extending the Orbit stack policy

Tail calls with multiple continuations:
\[(f \ e1 \ e2 \ k1 \ k2 \ k3)\]

In general, can't find youngest continuation statically.
At runtime, compare pointers of \(k1\), \(k2\), \(k3\) to \(sp\).
Extending the Orbit stack policy

Tail calls with multiple continuations:

\[(f \ e1 \ e2 \ k1 \ k2 \ k3)\]

In general, can’t find youngest continuation statically. At runtime, compare pointers of \(k1, k2, k3\) to \(sp\).
Continuation-Age (Cage) analysis

Possible solution:
compare ages of continuation closures that flow to call site.

(((λ(f k)
    ... (f "foo" clam₁ k) ...
    ... (f "bar" clam₂ clam₃) ...)
(λ(u k₁ k₂) call)
halt)
Continuation-Age (Cage) analysis

Possible solution:
compare ages of continuation closures that flow to call site.

\[
((\lambda(f \ k)
    \ldots (f \ "foo" \ clam_1 \ k) \ldots
    \ldots (f \ "bar" \ clam_2 \ clam_3) \ldots)
(\lambda(u \ k1 \ k2) \ call)
\]
\[\text{halt}\]

k1: \(clam_1, clam_2\)
k2: \(halt, clam_3\)
Continuation-Age (Cage) analysis

Possible solution:
compare ages of continuation closures that flow to call site.

$$((\lambda(f \ k)
  \ldots (f \ "foo" \ clam_1 \ k) \ldots
  \ldots (f \ "bar" \ clam_2 \ clam_3) \ldots)
(\lambda(u \ k1 \ k2) \ call)
halt)$$

$$clam_1 \preceq halt \quad \checkmark$$

$$clam_2 \preceq clam_3 \quad \checkmark$$

k1: $clam_1, \ clam_2$

k2: $halt, \ clam_3$
Continuation-Age (Cage) analysis

Possible solution: compare ages of continuation closures that flow to call site.

\[
(((\lambda(f\ k)
    \quad \ldots (f\ "\text{foo}\"\ clam_1\ k)\ \ldots
    \quad \ldots (f\ "\text{bar}\"\ clam_2\ clam_3)\ \ldots)
(\lambda(u\ k1\ k2)\ \text{call})
\text{halt})
\]

\[\begin{align*}
k1: & \quad \text{clam}_1, \text{clam}_2 \\
k2: & \quad \text{halt}, \text{clam}_3
\end{align*}\]

\[\begin{align*}
\text{clam}_1 \preceq \text{halt} & \quad \checkmark \\
\text{clam}_2 \preceq \text{clam}_3 & \quad \checkmark \\
\text{clam}_2 \preceq \text{halt} & \quad \checkmark \\
\text{clam}_1 \preceq \text{clam}_3 & \quad \times
\end{align*}\]
Cage analysis: take two

\[
((\lambda (f \ k)
\quad \ldots (f \ "\text{foo}\" \ clam_1 \ k) \ldots
\quad \ldots (f \ "\text{bar}\" \ clam_2 \ clam_3) \ldots)
(\lambda (u \ k1 \ k2) \text{call})
\text{halt})
\]

Better solution (possible by RCPS):

- Reason about continuation variables directly.
- Record total orders of continuation variables bound by the same user lambda.
Cage analysis: Ordering continuation variables

$$(\lambda(f\ k)$$

$$\ldots (f\ "foo"\ clam_1\ k)\ \ldots$$

$$\ldots (f\ "bar"\ clam_2\ clam_3)\ \ldots)$$

$$(\lambda(u\ k1\ k2)\ call)$$

$$(halt)$$

1st call  $k1 \preceq k2$
Cage analysis: Ordering continuation variables

\[
\left( \lambda (f \, k) \right.
\ 
\left( \ldots (f \ "foo" \ clam_1 \, k) \ldots \right.
\ 
\left( \ldots (f \ "bar" \ clam_2 \ clam_3) \ldots \right)
\ 
\left( \lambda (u \, k_1 \, k_2) \ call \right)
\ 
\text{halt} \right)
\]

1st call \quad k_1 \preceq k_2

2nd call \quad k_1 \preceq k_2

Overall \quad k_1 \preceq k_2
Cage analysis: Flowing age information

\((\lambda_1(u_1 k_1 k_2 k_3) \ldots (u_1 k_1 k_3 \text{ clam}_2 \text{ clam}_3) \ldots)\)

On entering \(\lambda_1\):

- \(\langle\{k_3\}, \{k_1\}, \{k_2\}\rangle\)
- \(u_1\) bound to \((\lambda_4(k_4 k_5 k_6 k_7)\text{ call})\)
Cage analysis: Flowing age information

\( (\lambda_1(u_1, k_1, k_2, k_3) \ldots (u_1, k_1, k_3, clam_2, clam_3) \ldots) \)

On entering \( \lambda_1 \):

- \( \langle \{k_3\}, \{k_1\}, \{k_2\} \rangle \)
- \( u_1 \) bound to \( (\lambda_4(k_4, k_5, k_6, k_7)\, call) \)

\( k_2 \) not used \( \langle \{k_3\}, \{k_1\} \rangle \)
Cage analysis: Flowing age information

\((\lambda_1(u1 \ k1 \ k2 \ k3)
\ldots (u1 \ k1 \ k3 \ clam_2 \ clam_3) \ldots)\)

On entering \(\lambda_1\):

- \(\langle\{k3\}, \{k1\}, \{k2\}\rangle\)
- \(u1\) bound to \((\lambda_4(k4 \ k5 \ k6 \ k7)\ call)\)

\(k2\) not used

\(clam_2, clam_3\) new

\(\langle\{k3\}, \{k1\}\rangle\)

\(\langle\{clam_2, clam_3\}, \{k3\}, \{k1\}\rangle\)
Cage analysis: Flowing age information

\((\lambda_1 (u1 \ k1 \ k2 \ k3))\)
\(\ldots (u1 \ k1 \ k3 \ clam_2 \ clam_3) \ldots)\)

On entering \(\lambda_1\):

- \(\langle\{k3\}, \{k1\}, \{k2\}\rangle\)
- \(u1\) bound to \((\lambda_4 (k4 \ k5 \ k6 \ k7) \text{call})\)

- \(k2\) not used \(\langle\{k3\}, \{k1\}\rangle\)
- \(clam_2, clam_3\) new \(\langle\{clam_2, clam_3\}, \{k3\}, \{k1\}\rangle\)
- actuals to formals \(\langle\{k6, k7\}, \{k5\}, \{k4\}\rangle\)
Also in the paper

- RCPS natural fit for multi-return lambda calculus.
- Multi-return lambda calculus $\xrightarrow{\text{CPS}}$ RCPS
- Implementation in Scheme48.
Evaluation

LALR parser in RCPS

184 multi-continuation calls (152 two-cont, 32 three-cont)
164 variable only
Evaluation

LALR parser in RCPS
184 multi-continuation calls (152 two-cont, 32 three-cont)
164 variable only

Cage with $k = 0$
142 resolved completely (87%)
22 resolved partially (ruled out one continuation)
Evaluation

LALR parser in RCPS
184 multi-continuation calls (152 two-cont, 32 three-cont)
164 variable only

Cage with $k = 0$
142 resolved completely (87%)
22 resolved partially (ruled out one continuation)

Control is less variant than data.
Conclusions

- Manage multi-continuation CPS with a stack.
- RCPS enables better reasoning about stack.
- Cage analysis to find youngest continuation statically.
Conclusions

- Manage multi-continuation CPS with a stack.
- RCPS enables better reasoning about stack.
- Cage analysis to find youngest continuation statically.

Thank you!