Delimited Continuations

The Bee’s Knees
Quasiconf 2012
Andy Wingo
A poll

How many of you use call/cc and continuation objects in large programs?

Do “we” really use it to implement coroutines and backtracking and threads and whatever?

Is call/cc necessary for Scheme?
Heresy

Those questions originally raised by racketeer Matthias Felleisen in 2000

Thesis of this presentation: call/cc bad, delimited continuations good
Against call/cc (1)

Requires set! to do almost anything with multiple returns

Passing arguments to continuations: manual CPS
Against call/cc (2)

“A global goto with arguments”

Captured continuations do not compose with current continuation:

(call/cc (lambda (k) (k (k 1))))

Oleg: “Call/cc is a bad abstraction.”
Against call/cc (3)

Delimited in practice...
...but where?
Almost always too much
Scheme deserves better

Delimited continuations

Sitaram 1993: “Handling Control”

Felleisen 1988: “The theory and practice of first-class prompts”
Bibliography, ctd

Flatt et al 2007: “Adding Delimited and Composable Control to a Production Programming Environment.”

Dybvig, Peyton-Jones, and Sabry 2007: “A monadic framework for delimited continuations”
Example.

(use-modules (ice-9 control))

(% (+ 1 (abort)) ; body
  (lambda (k) k)); handler

% pronounced "prompt"

What is captured:
(+ 1 [])

Wrapped in a function:
(lambda vals (+ 1 (apply values vals)))
Compositional

A function, not a global goto

(let ((k (% (+ 1 (abort))
            (lambda (k) k))))
  (k (k 1)))
= (((lambda vals (+ 1 (apply vals vals)))
   ((lambda vals (+ 1 (apply vals vals)))
    1))
  (+ 1 (+ 1 1))
= 3
Analogy with shell

fork/exec : coredump :: % : abort

Differences

~~ “Cores” from delimited continuations aren’t dead
~~ More expressive value passing
~~ Nestable
~~ The language, not the system
Tags

(% tag body handler)

(define-syntax-rule (let/ec k exp)
  (let (((tag (make-prompt-tag)))
    (% tag
      (let (((k (lambda args
                    (apply abort-to-prompt
                           tag
                           args)))))
        exp)
      (lambda (k . vals)
                (apply values vals))))))
Optimizations

Escape-only prompts

- Handler like `(lambda (k v ...) ...), k unreferenced`
- Implementable with setjmp/longjmp, no heap allocation
Optimizations

Prompt elision

- (% (make-prompt-tag) exp h) = exp
- Result of inlining (let/ec k body), k unreferenced in body
- Provide break, no cost if unused
Optimizations

Local CPS

Fundamentally dynamic: hence “dynamic control”
Mental model

Aborting to escape-only prompt: longjmp
Aborting to general prompt
- Copy of stack between prompt and abort
- Copy of dynamic bindings in same

Calling delimited continuation: splat stack, augment dynamic environment
Other names

“Composable continuations”
“Partial continuations”
Other formalisms

%/abort
%/control
call-with-prompt / abort-to-prompt
reset / shift
set / cupto
All equivalent
Limitations

Calling a delimited continuation composes two continuations: one stays in place, the other is pushed on

No way to use copying of C stack to do this: C stack frames are not relocatable

No standard way to capture continuation without unwinding to prompt
But what do I do with it?

A prompt is a boundary between programs
Prompts best conceived as concurrency primitives
The REPL and your code run concurrently
Node with automatic CPS

Delimited continuations: the ideal building block for lightweight threads
Set file descriptors to non-blocking
If EWOULDBLOCK, abort
Scheduler installs prompt, runs processes
(ice-9 nio)
nio-read
(ice-9 eports)

fdes->eport
file-port->eport
accept-eport
connect-eport
get-u8, etc
(ice-9 ethreads)

run

spawn, suspend, resume, sleep
(define (socket-loop esocket store)
  (let loop ()
    (let ((client (accept-eport esocket)))
      (spawn (lambda ()
                 (client-loop client store)))
             (loop))))
(define (client-loop eport store)
  (let loop ()
    (let* ((args (string-split
                  (read-line eport) #\space))
           (verb (string->symbol (car args)))
           (proc (hashq-ref *commands* verb)))
      (unless proc
        (client-error eport "Bad: ~a" verb)
        (proc eport store (cdr args)))
      (drain-output eport)
      (if (eof-object? (lookahead-u8 eport))
          (close-eport eport)
          (loop))))
BEES' KNEES?

YESS, SURE- BEES' KNEES. HE IS'N' FOND OF THEM.
questions?

- Guile: http://gnu.org/s/guile/
- Ethreads branch: wip-ethreads in Guile
- Words: http://wingolog.org/