

Compiling with Continuations

Matthew Might

University of Utah

matt.might.net

www.ucombinator.org

Administrivia

- Major update to course notes
- Final project after Thanksgiving

Today

- Exceptions from continuations
- Continuation-passing style, CPS
- How to compile continuations

History

- Plotkin, 1975
- Steele, 1978

Exceptions

- (try expression catch-procedure)
- (throw exception)

Exceptions

(dynamic-wind before-thunk
thunk
after-thunk)

Implementation

- Save and restore the stack + context
- Convert to continuation-passing style

Stack save/restore

- Pro: Works for most languages
- Con: Stacks are large; expensive

*context in C

- `setcontext()`
- `getcontext()`
- `makecontext()`
- `swapcontext()`

Continuation-passing style (CPS)

The CPS constraints

All calls are tail calls.

The CPS constraints

Function calls never return.

In practice

- All arguments must be atomic
- Atomic = must halt & be pure

CPS grammar

$v \in VAR ::= \textit{identifier}$
 $REF ::= v^\ell$
 $lam \in LAM ::= (\lambda (v_1 \cdots v_n) \textit{call})^\ell$
 $e, f \in EXP = REF + LAM$
 $call \in CALL ::= (f e_1 \cdots e_n)^\ell$

Intuition

- Callers pass current continuation to callees
- Callees invoke the continuation once done

Example

(lambda (x) x)

Example

`(lambda (x k) (k x))`

Example

```
(lambda (x return) (return x))
```

Example

```
(define (f n)
  (if (= n 0)
      1
      (* n (f (- n 1)))))
```

Example

```
(define (f n return)
  (if (= n 0)
      1
      (* n (f (- n 1)))))
```

Example

```
(define (f n return)
  (if (= n 0)
      (return 1)
      (* n (f (- n 1)))))
```

Example

```
(define (f n return)
  (if (= n 0)
      (return 1)
      (f (- n 1) (lambda (m)
                    (return (* n m))))))
```

Example

```
(define (f n return)
  (= $ n 0 (lambda (zero?)
    (if zero?
      (return 1)
      (- $ n 1 (lambda (sub)
        (f sub (lambda (mul)
          (* $ n mul return))))))))))
```

Example

```
(define (f a n return)
  (= $ n 0 (lambda (zero?)
    (if zero?
      (return a)
      (* a n (lambda (an)
        (- n 1 (lambda (n1)
          (f an n1 return))))))))))
```

Example

```
(define (fib n return)
  (<=$ n 0 (lambda (zero?)
    (if zero?
      (return n)
      (- n 1 (lambda (n1)
        (- n 2 (lambda (n2)
          (fib n1 (lambda (f1)
            (fib n2 (lambda (f2)
              (+ f1 f2))))))))))))))
```

CPS: All calls, no return

So why have a stack?

Procedure call

It's goto.

Translating

```
(+ 1 2 (lambda (a) ...))
```

```
a = 1 + 2 ;
```

Translating

```
(f x y cc)
```

```
$a1 = x
```

```
$a2 = y
```

```
$a3 = cc
```

```
goto f
```

Example

```
(define (fib n return)
  (<=$ n 0 (lambda (zero?)
    (if zero?
      (return n)
      (- n 1 (lambda (n1)
        (- n 2 (lambda (n2)
          (fib n1 (lambda (f1)
            (fib n2 (lambda (f2)
              (+ f1 f2))))))))))))))
```

```
fib: mov n $a1
      mov return $a2
      leq $t1 n 0
      bnz $t1 done
      sub n1 n 1
      sub n2 n 2
      ???
done: mov $a1 n
      goto return
```

Translating

```
(lambda (v1 ... vN) body)
```

```
(closure (lambda ($e v1 ... vN) body)  
         (make-env ...))
```

Translating

```
(closure (lambda ($e v1 ... vN) $body)
         (make-env ...))
```

```
{ lambda = &&label ,
  env   = { fv1 = ... } }
```

Compiling call/cc

call/cc

=>

```
(lambda (f current-continuation)
  (f (lambda (x later-continuation)
        (current-continuation x))
     current-continuation)))
```

Example

U

Example

map

In the wild

Web programming.

Holiday puzzle

(call/cc call/cc)