Continuation Passing Style

COMPSCI 631

University of Massachusetts Amherst

October 31, 2017

Recap: Implicit Continuations

The continuation of an expression is "what it should do next" after the expression is evaluated.

Syntax

$$
\begin{array}{rclr}
e & = & n & \text{numbers} \\
 & | & x & \text{identifiers} \\
 & | & e_1 + e_2 & \text{addition} \\
 & | & \textbf{fun } x \,.\, e & \text{functions} \\
 & | & e_1\, e_2 & \text{applications} \\
 & | & \boxed{\textbf{let } x = e_1 \textbf{ in } e_2} & \text{let bindings}
\end{array}
$$

Semantics

$$\text{E-NUM}\quad n \Downarrow n$$

$$\text{E-ADD}\quad \frac{e_1 \Downarrow m \qquad e_2 \Downarrow n}{e_1 + e_2 \Downarrow m + n}$$

$$\text{E-APP}\quad \frac{e_1 \Downarrow \textbf{fun } x \,.\, e \qquad e_2 \Downarrow v \qquad e[\, x \,/\, v] \Downarrow v'}{e_1\, e_2 \Downarrow v'}$$

$$\text{E-FUN}\quad \textbf{fun } x \,.\, e \Downarrow \textbf{fun } x \,.\, e$$

$$\text{E-LET}\quad \frac{e_1 = \Downarrow v1 \qquad e_2[x/v1] = \Downarrow v2}{\textbf{let } x = e_1 \textbf{ in } e_2 = \Downarrow v2}$$

A "stack" in a proof

$$\cfrac{\cfrac{1+4 \Downarrow 5}{(1+4) + 3 \Downarrow 8}}{((1+4)+3) + 7 \Downarrow 15}$$

Recap: Explicit Continuations

$$\kappa ::= \textbf{top} \mid \textbf{add}_R(e_2,\ \kappa) \mid \textbf{add}_L(\mathsf{m},\ \kappa) \mid \textbf{app}_R(e_2,\ \kappa) \mid \textbf{app}_L(\mathsf{x},\ e,\ \kappa) \mid \textbf{let}(\mathsf{x}, e_2, \kappa)$$

$e_1 + e_2,\ \kappa \rightarrow e_1,\ \textbf{add}_R(e_2,\ \kappa)$

$\mathsf{m},\ \textbf{add}_R(e_2,\ \kappa) \rightarrow e_2,\ \textbf{add}_L(\mathsf{m},\ \kappa)$

$\mathsf{m} + \mathsf{n},\ \kappa \rightarrow \mathsf{r},\ \kappa$ where $r = m + n$

$e_1\, e_2, \kappa \rightarrow e_1, \textbf{app}_R(e_2, \kappa)$

$\textbf{fun}\ \mathsf{x}\ .\ e, \textbf{app}_R(e_2, \kappa) \rightarrow e_2, \textbf{app}_L(\mathsf{x},\ e,\ \kappa)$

$v, \textbf{app}_L(\mathsf{x},\ e,\ \kappa) \rightarrow e[\mathsf{x}/v], \kappa$

$\textbf{let}\ \mathsf{x} = e_1\ \textbf{in}\ e_2, \kappa \rightarrow e_1, \textbf{let}(\mathsf{x}, e_2, \kappa)$

$v, \textbf{let}(\mathsf{x}, e_2, \kappa) \rightarrow e_2[\mathsf{x}/v], \kappa$

$e_1, \kappa \rightarrow e_2, \kappa'$ is a single step. We need to apply the step repeatedly until we get $v, \textbf{top}$.

Note: We still have a "stack" (i.e., κ). The stack is simply an explicit data structure.

Recap: Explicit Continuations

$$\kappa ::= \textbf{top} \mid \textbf{add}_R(e_2, \kappa) \mid \textbf{add}_L(m, \kappa) \mid \textbf{app}_R(e_2, \kappa) \mid \textbf{app}_L(x, e, \kappa) \mid \textbf{let}(x, e_2, \kappa)$$

$e_1 + e_2,\ \kappa \rightarrow e_1,\ \textbf{add}_R(e_2,\ \kappa)$

$m,\ \textbf{add}_R(e_2,\ \kappa) \rightarrow e_2,\ \textbf{add}_L(m,\ \kappa)$

$m + n,\ \kappa \rightarrow r,\ \kappa$ where $r = m + n$

$e_1\, e_2, \kappa \rightarrow e_1, \textbf{app}_R(e_2, \kappa)$

$\textbf{fun}\ x\ .\ e, \textbf{app}_R(e_2, \kappa) \rightarrow e_2, \textbf{app}_L(x,\ e,\ \kappa)$

$v, \textbf{app}_L(x,\ e,\ \kappa) \rightarrow e[x/v], \kappa$

$\textbf{let}\ x = e_1\ \textbf{in}\ e_2, \kappa \rightarrow e_1, \textbf{let}(x, e_2, \kappa)$

$v, \textbf{let}(x, e_2, \kappa) \rightarrow e_2[x/v], \kappa$

$e_1, \kappa \rightarrow e_2, \kappa'$ is a single step. We need to apply the step repeatedly until we get $v, \textbf{top}$.

Note: We still have a “stack” (i.e., κ). The stack is simply an explicit data structure.

A trivial optimization: $(\textbf{fun}\ x\ .\ e)\ v,\ \kappa \rightarrow e[x/v], \kappa$

Key Idea: We are going to transform e into an equivalent program e', such that the only continuations it uses are:

1. $\kappa = \textbf{top}$, i.e., *nothing to do next*
2. $\kappa = \textbf{let}(x, e, \textbf{top})$, i.e., *name then current value x and then evaluate e*

Note that both κs have fixed depth (0 or 1). So, we effectively do not use the stack.

Continuation Passing Style

This expression uses the stack so we cannot write it:

$(1 + 2) + 3$

However, we can rewrite it to:

let $x = 1 + 2$ **in** $x + 3$

Continuation Passing Style

This expression uses the stack so we cannot write it:

(1 + 2) + 3

However, we can rewrite it to:

let x = 1 + 2 **in** x + 3

This expression uses the stack so we cannot write it:

let f = **fun** x . x + 1 **in**
let g = **fun** y . f y + 20 **in**
g 300

However, we can rewrite it to:

let f = **fun** x . **fun** k . **let** r = x + 1 **in** k r **in**
let g = **fun** y . **fun** k . f y (**fun** r0 . **let** r1 = r0 + 20 **in** k r1)
g 300 (**fun** r . r)

High-level idea: Instead of returning a value, every function takes an extra argument *k* that receives the value the original function would have returned.
Verify that the stack is no longer used.

Compiling to Continuation Passing Style

In OCaml