Continuations

COMPSCI 631

University of Massachusetts Amherst

October 31, 2017

Some applications of continuations

1. Write an interpreter using loops instead of recursion.
2. Compile a functional language to assembly.
3. Write a "logic programming language", such as Prolog.
4. Transliterate an interpreter from OCaml to Haskell while preserving call-by-value semantics.
5. Implement threads without using hardware threads.
6. Run a computationally expensive program in the browser without locking up the user-interface.
7. Write a programming language that allows a single program to transparently run on a web server and a web browser. i.e., the language handles all communication transparently.

What is a continuation?

The continuation of an expression is "what it should do next" after the expression is evaluated. In the interpreters you have written, when an expression e is reduced to a value v, the answer to the question "what to do next" is "return the value v and keep running the interpreter". i.e., the machine's stack describes "what to do next".

What is a continuation?

The continuation of an expression is "what it should do next" after the expression is evaluated. In the interpreters you have written, when an expression e is reduced to a value v, the answer to the question "what to do next" is "return the value v and keep running the interpreter". i.e., the machine's stack describes "what to do next".

Syntax

$$
\begin{array}{rcll}
e & = & n & \text{numbers} \\
 & | & x & \text{identifiers} \\
 & | & e_1 + e_2 & \text{addition} \\
 & | & \textbf{fun}\ \mathsf{x}\ .\ e & \text{functions} \\
 & | & e_1\, e_2 & \text{applications}
\end{array}
$$

Semantics

$$\textsc{E-Num}\quad \mathsf{n} \Downarrow \mathsf{n}$$

$$\textsc{E-Add}\quad \frac{e_1 \Downarrow \mathsf{m} \qquad e_2 \Downarrow \mathsf{n}}{e_1 + e_2 \Downarrow \mathsf{m} + \mathsf{n}}$$

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

$$\textsc{E-Fun}\quad \textbf{fun}\ \mathsf{x}\ .\ e \Downarrow \textbf{fun}\ \mathsf{x}\ .\ e$$

What is a continuation?

The continuation of an expression is "what it should do next" after the expression is evaluated. In the interpreters you have written, when an expression e is reduced to a value v, the answer to the question "what to do next" is "return the value v and keep running the interpreter". i.e., the machine's stack describes "what to do next".

Syntax

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

Semantics

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

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

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

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

A "stack" in a proof

$$\frac{\dfrac{1{+}4 \Downarrow 5 \qquad 3 \Downarrow 3}{(1{+}4) + 3 \Downarrow 8} \qquad 7 \Downarrow 7}{((1{+}4){+}3) + 7 \Downarrow 15}$$

Semantics with an explicit continuation

Syntax

$$
\begin{array}{llll}
e & = & n & \text{numbers} \\
 & | & x & \text{identifiers} \\
 & | & e_1 + e_2 & \text{addition} \\
 & | & \textbf{fun}\ \mathsf{x}\ .\ e & \text{functions} \\
 & | & e_1\, e_2 & \text{applications}
\end{array}
$$

Original Semantics

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

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

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

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

$$
\begin{array}{lll}
\kappa & ::= & \textbf{top} \\
 & | & \textbf{add}_R(e_2,\ \kappa) \\
 & | & \textbf{add}_L(\mathsf{m},\ \kappa) \\
 & | & \textbf{app}_R(e_2,\ \kappa) \\
 & | & \textbf{app}_L(\mathsf{x},\ e,\ \kappa)
\end{array}
$$

Semantics with Explicit Continuations

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

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

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

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

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

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

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

Semantics with an explicit continuation

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

$$
\begin{aligned}
& e_1 + e_2,\ \kappa \rightarrow e_1,\ \textbf{add}_R(e_2,\ \kappa)\\
& \mathsf{m},\ \textbf{add}_R(e_2,\ \kappa) \rightarrow e_2,\ \textbf{add}_L(\mathsf{m},\ \kappa)\\
& \mathsf{m} + \mathsf{n},\ \kappa \rightarrow \mathsf{r} \text{ where } r = m + n\\
& e_1\, e_2, \kappa \rightarrow e_1, \textbf{app}_R(e_2, \kappa)\\
& \textbf{fun}\ \mathsf{x}\ .\ e, \textbf{app}_R(e_2, \kappa) \rightarrow e_2, \textbf{app}_L(\mathsf{x},\ e,\ \kappa)\\
& v, \textbf{app}_L(\mathsf{x},\ e,\ \kappa) \rightarrow e[\mathsf{x}/v], \kappa
\end{aligned}
$$

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

Some observations

1. Since κ is a data structure, we can store it, send it on the network, etc.
2. Since $e_1, \kappa \rightarrow e_2, \kappa'$ is a single step, we can pause computation and resume it later (or never resume it).