# CPS Transformation of Lisp-Like Multi-Staged Languages

Ludovic Patey

8 juillet 2011

# Table of contents

# CPS Transformation

Continuation Passing Style : continuation as argument

- Normal version

```
let plus a b = a + b;;

print_int (plus 3 2)
```

- CPS version

```
let plus a b k = k (a + b);;

plus 3 2 print_int
```

# CPS Transformation

## Indifference

$$Eval_N(\underline{e}\ \ (\lambda x.x)) = Eval_V(\underline{e}\ \ (\lambda x.x))$$

- $Eval_V$ is a Call-By-Value evaluator
- $Eval_N$ is a Call-By-Name evaluator

# Applications

- Uniform reasoning about continuations
  - eg. Compiler optimizations
- Indifference of evaluation strategy
  - eg. Program verification

# CPS Transformation

## Fischer & Plotkin

(TVAR) $$x \mapsto \lambda k.\,(k\ x)$$

(TABS) $$\frac{e \mapsto \underline{e}}{\lambda x.e \mapsto \lambda k.\,(k\ \lambda x.\underline{e})}$$

(TAPP) $$\frac{e_1 \mapsto \underline{e_1} \qquad e_2 \mapsto \underline{e_2}}{e_1\ e_2 \mapsto \lambda k.\underline{e_1}\ \left(\lambda m.\underline{e_2}\,(\lambda n.\,(m\ n\ k))\right)}$$

# Multi-Staged Calculus

## Syntax

$$\textit{Expr} \quad e ::= i \mid x \mid \lambda x.e \mid e\ e \mid \textbf{let } x = e \textbf{ in } e \mid \textbf{box } e \mid \textbf{run } e \mid \textbf{unbox} \mid \textbf{lift } e$$

$$\begin{aligned} \textbf{run } (\textbf{box } e) &= e \\ \textbf{unbox } (\textbf{box } e) &= e \\ \text{if } e \rightsquigarrow v \text{ then } \textbf{lift } e &= \textbf{box } v \end{aligned}$$

# Multi-Staged Calculus

**Example 1**

$$\textbf{let } x = 1 \textbf{ in } (\textbf{box } x) \rightsquigarrow \textbf{box } x$$

**Example 2**

$$\textbf{run } (\textbf{box } x) \rightsquigarrow \text{ERROR}$$

# Multi-Staged Calculus

## Example 3

$$\begin{array}{ll} \textbf{let} & a = \textbf{box}\ x \\ & b = \textbf{box}\ (\lambda x.\lambda y.(\textbf{unbox}\ a) + y) \\ \textbf{in} & (\textbf{run}\ b)\ 1\ 1 \end{array}$$

$$\rightsquigarrow \begin{array}{ll} \textbf{let} & b = \textbf{box}\ (\lambda x.\lambda y.(\textbf{unbox}\ (\textbf{box}\ x)) + y) \\ \textbf{in} & (\textbf{run}\ b)\ 1\ 1 \end{array}$$

$$\rightsquigarrow \begin{array}{ll} \textbf{let} & b = \textbf{box}\ (\lambda x.\lambda y.x + y) \\ \textbf{in} & (\textbf{run}\ b)\ 1\ 1 \end{array}$$

# CPS Transformation

**Definitions**

$$\begin{array}{ll} \textit{Context} & \kappa ::= (e\ \lambda h.[\cdot])\ \mid\ (e\ \lambda h.\kappa) \\ \textit{Context stack} & K ::= \bot \mid K, \kappa \end{array}$$

**Transformation**

$$(\texttt{TVAR}) \qquad x \mapsto (\lambda k.\,(k\ x)\,,\bot)$$

$$(\texttt{TABS}) \qquad \frac{e \mapsto (\underline{e}, K)}{\lambda x.e \mapsto (\lambda k.\,(k\ \lambda x.\underline{e})\,, K)}$$

$$(\texttt{TAPP}) \qquad \frac{e_1 \mapsto (\underline{e_1}, K_1) \qquad e_2 \mapsto (\underline{e_2}, K_2)}{e_1\ e_2 \mapsto \left(\lambda k.\underline{e_1}\ \left(\lambda m.\underline{e_2}\,(\lambda n.\,((m\ n)\ \ k))\right)\,, K_1 \bowtie K_2\right)}$$

# CPS Transformation

(TBOX)
$$\frac{e \mapsto (\underline{e}, (K, \kappa))}{\textbf{box } e \mapsto (\lambda k.\kappa\,[k\ (\textbf{box } \underline{e})]\,, K)}$$

$$\frac{e \mapsto (\underline{e}, \bot)}{\textbf{box } e \mapsto (\lambda k.k\ (\textbf{box } \underline{e})\,, \bot)}$$

(TUNB)
$$\frac{e \mapsto (\underline{e}, K)}{\textbf{unbox } e \mapsto (\textbf{unbox } h, (K, \underline{e}\ (\lambda h.[\cdot])))}$$

(TLIF)
$$\frac{e \mapsto (\underline{e}, K)}{\textbf{lift } e \mapsto (\lambda k.(\underline{e}\ (\lambda m.k\ (\textbf{lift}\ (\lambda n.n\ m))))\,, K)}$$

(TRUN)
$$\frac{e \mapsto (\underline{e}, K)}{\textbf{run } e \mapsto (\lambda k.(\underline{e}\ (\lambda m.((\textbf{run}\ m)\ k)))\,, K)}$$

# Conclusion

- CPS is extendable to Multi-Staged Calculus
- It enables direct compilation optimizations on Multi-Staged languages.
- Is it necessary ? (Sabry)

# Références

I.S. Kim, K. Yi, and C. Calcagno.
A polymorphic modal type system for lisp-like multi-staged languages.
In *Conference Record of the ACM Symposium on Principles of Programming Languages*, volume 33, page 257. Citeseer, 2006.

G.D. Plotkin.
Call-by-name, call-by-value and the [lambda]-calculus.
*Theoretical computer science*, 1(2) :125–159, 1975.