Explicit Polymorphism and CPS Conversion

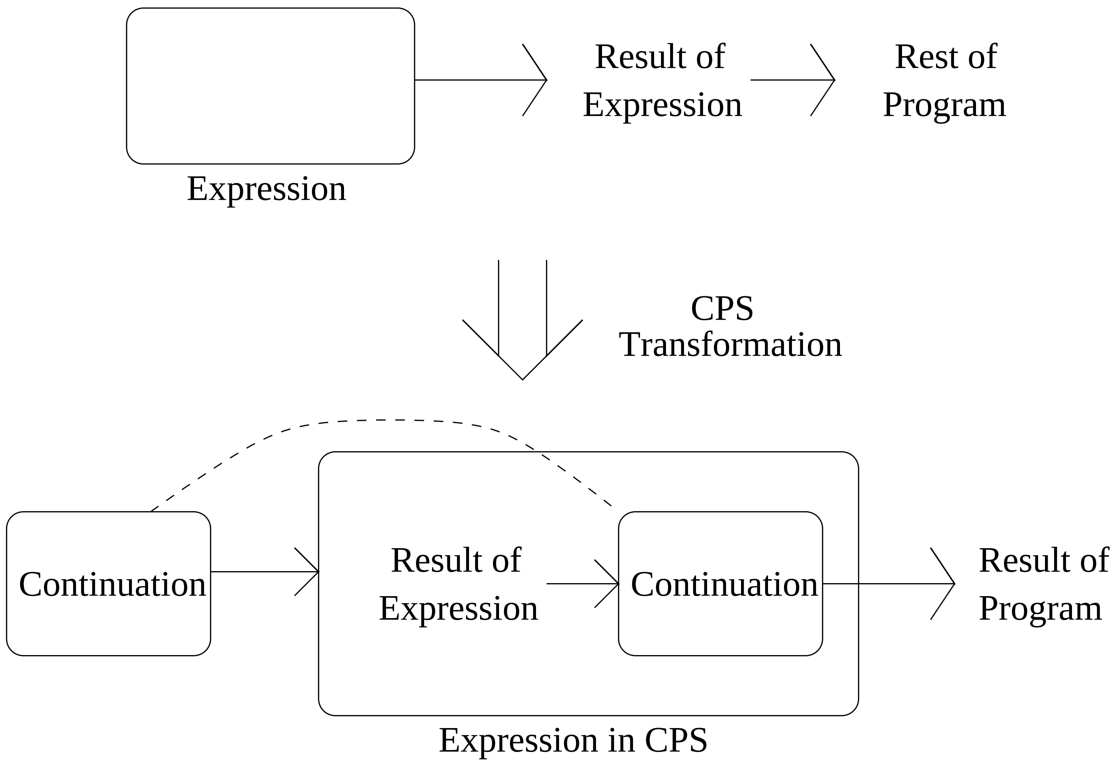
Robert Harper, Mark Lillibridge
Presented by JungTaek Kim

Overview

- Explain CPS and F_ω.
- How we can assign types for CPS terms?
- (Actually, it talks about several semantics and control operators)

About CPS Transformation

- CPS (Continuation Passing Style) Transformation transforms all expression to receive a continuation (remaining computations).



CPS Trans. for λ calculus

- λ calculus language

$$\begin{array}{rcll} e & ::= & x & \text{variable} \\ & | & \lambda x.e & \text{abstraction} \\ & | & e_1 @ e_2 & \text{application} \end{array}$$

- CPS Transformation

$$\begin{array}{rcl} \langle x \rangle & = & \lambda k.k @ x \\ \langle \lambda x.e \rangle & = & \lambda k.k @ \lambda x.\langle e \rangle \\ \langle e_1 @ e_2 \rangle & = & \lambda k.\langle e_1 \rangle @ \lambda f.(\langle e_2 \rangle @ \lambda v.(f @ v @ k)) \end{array}$$

F_ω

- This can be thought as explicit-typed and explicit-polymorphism λ calculus.

- Terms

$$
\begin{array}{lcll}
e & ::= & x & \text{variable} \\
 & | & \lambda x{:}\alpha.e & \text{abstraction} \\
 & | & e_1 @ e_2 & \text{application} \\
 & | & \Lambda t{:}K.e & \text{type abstraction} \\
 & | & e_1 @ \{\alpha\} & \text{type application}
\end{array}
$$

- Constructors (Type with computation)

$$
\begin{array}{lcll}
\alpha & ::= & \beta & \text{base type} \\
 & | & u & \text{type variable} \\
 & | & \alpha \to \alpha & \text{type variable} \\
 & | & \forall u{:}K.\alpha & \text{polymorphic type} \\
 & | & \lambda u{:}K.\alpha & \text{existential type} \\
 & | & \alpha_1 @ \alpha_2 & \text{type specializtion}
\end{array}
$$

- Kinds

$$
\begin{array}{lcll}
K & ::= & \Omega & \text{kind for types} \\
 & | & K_1 \Rightarrow K_2 & \text{kind for polymorphic} \\
 & & & \text{and existential type}
\end{array}
$$

Example in F_ω

- Type of polymorphic function.

  In ML, `fn x => x` and `(fn x => x) 1`

  And in F_ω, $\Lambda t{:}\Omega.\lambda x{:}t.x$ and $(\Lambda t{:}\Omega.\lambda x{:}t.x)@\{int\}@1$

- Polymorphic function which apply a function with an argument.

  In ML, `fn f => fn x => f x`

  And in F_ω,

$$
\begin{array}{lll}
\Lambda t{:}\Omega. & & \\
& \lambda f{:}(\forall t'{:}\Omega.t' \to t'). & \\
& & \lambda x{:}t.\ f@\{t\}@x
\end{array}
$$

Standard and Call-by-value Semantics of F_ω

- Use evaluation context style semantics.
- Define values, redices and contexts.

$$\begin{array}{rcl} v & ::= & x \\ & \mid & \lambda x{:}\alpha.e \\ & \mid & \Lambda u{:}K.e \\ \\ r & ::= & (\lambda x{:}\alpha.e)@v \\ & \mid & (\Lambda u{:}K.e)@\{\alpha\} \\ \\ C & ::= & [] \\ & \mid & C@e \\ & \mid & v@C \\ & \mid & C@\{\alpha\} \end{array}$$

- Reductions.

$$\begin{array}{rcl} C[(\lambda x{:}\alpha.e)@v] & \hookrightarrow & C[[v/x]e] \\ C[(\Lambda u{:}K.e)@\{\alpha\}] & \hookrightarrow & C[[\alpha/u]e] \end{array}$$

- Why standard?

  Type abstractions are really values and type applications are really computations.

What are the types for CPS forms

- Continuation types

  Continuation receive the values and return answer of "entire" program.

  Let's the type of "entire" program be ans.

  Then the type of continuation will be :

  $$\alpha \to ans$$

  for some type α.

- Example types for CPS function values.

$$\begin{array}{rcl} \beta \to \beta & \Rightarrow_{cps} & \beta \to (\beta \to ans) \to ans \\ (\beta \to \beta) \to \beta & \Rightarrow_{cps} & (\beta \to (\beta \to ans) \to ans) \\ & & \to (\beta \to ans) \to ans \end{array}$$

Transformation of types for CPS terms : λ calculus

- Transformation of types can be defined using two mutually recursive functions $|\cdot|$ and $\|\cdot\|$.

$$|\alpha| \equiv (\|\alpha\| \to ans) \to ans$$

$$\begin{aligned} \|\beta\| &\equiv \beta \\ \|u\| &\equiv u \\ \|\alpha_1 \to \alpha_2\| &\equiv \|\alpha_1\| \to |\alpha_2| \end{aligned}$$

- Above example.

$$\beta \to \beta \quad \Rightarrow_{cps} \quad \beta \to (\beta \to ans) \to ans \to ans$$

Typed CPS transformation : λ calculus

- Assume that type checking is completed.
- $\langle e{:}\alpha \rangle \equiv e'$

$$
\begin{array}{rcl}
\langle x{:}\alpha \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).k@x \\
\langle (\lambda x{:}\alpha.e){:}\alpha \to \alpha' \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).k@(\lambda x{:}\|\alpha\|.\langle e \rangle) \\
\langle (e_1{:}\alpha_1)@(e_2{:}\alpha_2){:}\alpha \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).\langle e_1 \rangle @ \\
 & & \quad \lambda f{:}\|\alpha_2 \to \alpha\|.\langle e_2 \rangle @ \\
 & & \quad\quad \lambda v{:}\|\alpha_2\|.f@v@k
\end{array}
$$

Examples(CPS - λ)

Following λ-expression :

$$(\lambda x.x)@1$$

will be transformed to :

Transformation of types for CPS terms : F_ω

- Transformation of types can be defined using two mutually recursive functions $|\cdot|$ and $\|\cdot\|$.

$$
\begin{array}{rcl}
|\alpha| & \equiv & (\|\alpha\| \rightarrow ans) \rightarrow ans \\
\\
\|\beta\| & \equiv & \beta \\
\|u\| & \equiv & u \\
\|\alpha_1 \rightarrow \alpha_2\| & \equiv & \|\alpha_1\| \rightarrow |\alpha_2| \\
\|\forall u{:}K.\alpha\| & \equiv & \forall u{:}K.|\alpha| \\
\|\lambda u{:}K.\alpha\| & \equiv & \lambda u{:}K.\|\alpha\| \\
\|\alpha_1 @ \alpha_2\| & \equiv & \|\alpha_1\| @ \|\alpha_2\|
\end{array}
$$

Typed CPS transformation : F_ω

- Assume that type checking is completed.
- $\langle e{:}\alpha \rangle \equiv e'$

$$
\begin{array}{rcl}
\langle x{:}\alpha \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).k@x \\
\langle (\lambda x{:}\alpha.e){:}\alpha \to \alpha' \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).k@(\lambda x{:}\|\alpha\|.\langle e \rangle) \\
\langle (\Lambda u{:}K.e){:}\forall u{:}K.\alpha \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).k@(\Lambda u{:}K.\langle e \rangle) \\
\langle (e_1{:}\alpha_1)@(e_2{:}\alpha_2){:}\alpha \rangle & \equiv & \lambda k{:}(\|\alpha\| \to ans).\langle e_1 \rangle @ \\
& & \quad \lambda f{:}\|\alpha_2 \to \alpha\|.\langle e_2 \rangle @ \\
& & \qquad \lambda v{:}\|\alpha_2\|.f@v@k \\
\langle e@\{\alpha_1\}{:}[\alpha_1/u]\alpha_2 \rangle & \equiv & \lambda k{:}(\|[\alpha_1/u]\alpha_2\| \to ans).\langle e \rangle @ \\
& & \quad (\lambda v{:}\|\forall u{:}K_1.\alpha_2\|.v@\{\|\alpha_1\|\}@k)
\end{array}
$$