Polymorphic Delimited Continuations

Kenichi Asai — Ochanomizu University
Yukiyoshi Kameyama — University of Tsukuba

November 30, 2007

Outline of the talk

1. The language: λ-calculus + let + shift/reset
2. Example: type-safe printf (sub Main Part)
   - delimited continuations
   - answer type modification
   - monomorphic type system
3. Let-polymorphism (Main Part)
   - answer type polymorphism for delimited continuations
   - purity restriction
4. Correspondence to CPS translation
5. Extension to System F
6. Related Work / Conclusion

Message: Polymorphism can be introduced naturally.

The Language: λ-calculus + let + shift/reset

Syntax:

$$
\begin{array}{rcll}
v & ::= & x & \text{variable} \\
  & | & \lambda x.\, e & \text{abstraction} \\
e & ::= & v & \text{value} \\
  & | & e_1\, e_2 & \text{application} \\
  & | & \text{let}\, x = e_1 \,\text{in}\, e_2 & \text{let} \\
  & | & \mathcal{S} k.\, e & \text{shift} \\
  & | & \langle e \rangle & \text{reset}
\end{array}
$$

Evaluation context:

$$
\begin{array}{rcl}
E & ::= & [\,] \mid v\, E \mid E\, e \mid \langle E \rangle \\
  & | & \text{let}\, x = E \,\text{in}\, e
\end{array}
$$

Pure evaluation context:

$$
\begin{array}{rcl}
F & ::= & [\,] \mid v\, F \mid F\, e \\
  & | & \text{let}\, x = F \,\text{in}\, e
\end{array}
$$

(i.e., no reset surrounds the hole.)

Reduction Rules:

$E[R] \leadsto E[e]$ where $R \leadsto e$ is one of:

$$
\begin{array}{rcl}
(\lambda x.\, e)\, v & \leadsto & e[v/x] \\
\text{let}\, x = v \,\text{in}\, e & \leadsto & e[v/x] \\
\langle v \rangle & \leadsto & v \\
\langle F[\mathcal{S} k.\, e] \rangle & \leadsto & \langle \text{let}\, k = \lambda x.\, \langle F[x] \rangle \,\text{in}\, e \rangle
\end{array}
$$

(In the paper, we also have fix and if.)

Example: Type-safe Printf

Goal:

$$\begin{array}{ll} & \texttt{sprintf}\ (\lambda\_.\ \texttt{"Hello world!"}) \\ \rightsquigarrow & \texttt{"Hello world!"} \end{array}$$

$$\begin{array}{ll} & \texttt{sprintf}\ (\lambda\_.\ \texttt{"t is "}\ \texttt{\^{}}\ \texttt{\%}\ \texttt{int}\ \texttt{\^{}}\ \texttt{"!"})\ 4 \\ \rightsquigarrow & \texttt{"t is 4!"} \end{array}$$

$$\begin{array}{ll} & \texttt{sprintf}\ (\lambda\_.\ \texttt{\%}\ \texttt{str}\ \texttt{\^{}}\ \texttt{" is "}\ \texttt{\^{}}\ \texttt{\%}\ \texttt{int}\ \texttt{\^{}}\ \texttt{"!"})\ \texttt{"x"}\ 3 \\ \rightsquigarrow & \texttt{"x is 3!"} \end{array}$$

Solution:

$$\begin{array}{l} \texttt{let int} = \lambda x.\, \texttt{string\_of\_int}\, x \quad (= \texttt{string\_of\_int}) \\ \texttt{let str} = \lambda x.\, x \end{array}$$

$$\begin{array}{l} \texttt{let}\ \%\ \mathit{to\_str} = \mathcal{S}k.\, \lambda n.\, k\,(\mathit{to\_str}\ n) \\ \texttt{let sprintf}\ \mathit{thunk} = \langle \mathit{thunk}\,()\rangle \end{array}$$

cf. CPS solution: [Danvy '98]

Example: Type-safe Printf

Key observation:
(% int) changes the type of the context from a string to a function.

$$
\begin{array}{ll}
 & \texttt{sprintf}\ (\lambda\_.\ \texttt{"t is "}\ \texttt{\^{}}\ \%\ \texttt{int}\ \texttt{\^{}}\ \texttt{"!"})\ 4 \\
\rightsquigarrow & \langle(\lambda\_.\ \texttt{"t is "}\ \texttt{\^{}}\ \%\ \texttt{int}\ \texttt{\^{}}\ \texttt{"!"})\ ()\rangle\ 4 \\
\rightsquigarrow & \langle\texttt{"t is "}\ \texttt{\^{}}\ \%\ \texttt{int}\ \texttt{\^{}}\ \texttt{"!"}\rangle\ 4 \\
 & \\
\rightsquigarrow & \langle\texttt{"t is "}\ \texttt{\^{}}\ (\mathcal{S}k.\ \lambda n.\ k\ (\texttt{int}\ n))\ \texttt{\^{}}\ \texttt{"!"}\rangle\ 4 \\
\rightsquigarrow & \langle \mathrm{let}\ k = \lambda x.\ \langle\texttt{"t is "}\ \texttt{\^{}}\ x\ \texttt{\^{}}\ \texttt{"!"}\rangle\ \mathrm{in}\ \lambda n.\ k\ (\texttt{int}\ n)\rangle\ 4 \\
 & \\
\rightsquigarrow & \langle\lambda n.\ \langle\texttt{"t is "}\ \texttt{\^{}}\ \texttt{int}\ n\ \texttt{\^{}}\ \texttt{"!"}\rangle\rangle\ 4 \\
\rightsquigarrow & (\lambda n.\ \langle\texttt{"t is "}\ \texttt{\^{}}\ \texttt{int}\ n\ \texttt{\^{}}\ \texttt{"!"}\rangle)\ 4 \\
\rightsquigarrow & \langle\texttt{"t is "}\ \texttt{\^{}}\ \texttt{int}\ 4\ \texttt{\^{}}\ \texttt{"!"}\rangle \\
\rightsquigarrow & \langle\texttt{"t is 4!"}\rangle \\
\rightsquigarrow & \texttt{"t is 4!"} \qquad\qquad \text{cf. } \texttt{int} = \texttt{string\_of\_int}
\end{array}
$$

$$
\langle F[\mathcal{S}k.\ e]\rangle \quad \rightsquigarrow \quad \langle \mathrm{let}\ k = \lambda x.\ \langle F[x]\rangle\ \mathrm{in}\ e\rangle
$$

Answer Type Modification

Superscripts show the type of the expression.
i stands for int and s stands for string.

$$\langle \texttt{"t is "} \mathbin{\hat{}} \texttt{\% int} \mathbin{\hat{}} \texttt{"!"} \rangle^{\texttt{s}}\ 4$$

$$\begin{aligned}
&\rightsquigarrow \langle \texttt{"t is "} \mathbin{\hat{}} (\mathcal{S}k.\, \lambda n.\, k\, (\texttt{int}\, n))^{\texttt{s}} \mathbin{\hat{}} \texttt{"!"} \rangle^{\texttt{s}}\ 4 \\
&\rightsquigarrow \langle \text{let } k^{\texttt{s}\to\texttt{s}} = \lambda x.\, \langle \texttt{"t is "} \mathbin{\hat{}} x^{\texttt{s}} \mathbin{\hat{}} \texttt{"!"} \rangle^{\texttt{s}} \text{ in } \lambda n^{\texttt{i}}.\, k\, (\texttt{int}\, n) \rangle^{\texttt{i}\to\texttt{s}}\ 4 \\
&\rightsquigarrow \langle \lambda n^{\texttt{i}}.\, \langle \texttt{"t is "} \mathbin{\hat{}} (\texttt{int}\, n)^{\texttt{s}} \mathbin{\hat{}} \texttt{"!"} \rangle^{\texttt{s}} \rangle^{\texttt{i}\to\texttt{s}}\ 4
\end{aligned}$$

or more in general:

$$\langle F[\mathcal{S}k.\, e^{\beta}]^{\tau} \rangle^{\alpha} \quad \rightsquigarrow \quad \langle \text{let } k^{\tau\to\alpha} = \lambda x.\, \langle F[x]^{\tau} \rangle^{\alpha} \text{ in } e^{\beta} \rangle^{\beta}$$

Thus, a type judgement needs three types: $\boxed{\Gamma; \alpha \vdash e : \tau; \beta.}$

"Under a type environment Γ, an expression e has type τ, and the type of context (or answer type) changes from α to β."

Monomorphic Type System [DanvyFilinski '89]

Two Judgements: $\boxed{\Gamma \vdash_p e : \tau}$ and $\boxed{\Gamma; \alpha \vdash e : \tau; \beta}$.

Typing rules:

$$\frac{\Gamma \vdash_p e : \tau}{\Gamma; \alpha \vdash e : \tau; \alpha}\ \text{exp} \qquad \frac{(x : \tau \in \Gamma)}{\Gamma \vdash_p x : \tau}\ \text{var} \qquad \frac{\Gamma, x : \sigma; \alpha \vdash e : \tau; \beta}{\Gamma \vdash_p \lambda x.\, e : \sigma/\alpha \to \tau/\beta}\ \text{fun}$$

$$\frac{\Gamma; \gamma \vdash e_1 : \sigma/\alpha \to \tau/\beta; \delta \quad \Gamma; \beta \vdash e_2 : \sigma; \gamma}{\Gamma; \alpha \vdash e_1\, e_2 : \tau; \delta}\ \text{app}$$

$$\frac{\Gamma, k : \tau/t \to \alpha/t; \sigma \vdash e : \sigma; \beta}{\Gamma; \alpha \vdash \mathcal{S}k.\, e : \tau; \beta}\ \text{shift} \qquad \frac{\Gamma; \sigma \vdash e : \sigma; \tau}{\Gamma \vdash_p \langle e \rangle : \tau}\ \text{reset}$$

- The judgement $\Gamma; \alpha \vdash e : \tau; \beta$ and a function type $\sigma/\alpha \to \tau/\beta$ accommodate answer type modification.
- If no control effect is used, the answer type is always the same.

Example: Type-safe Printf

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{k : \mathtt{s} \to \mathtt{s} \vdash_p k : \mathtt{i} \to \mathtt{s}}\;\text{var}
\qquad
\dfrac{\cdots}{n : \mathtt{i} \vdash_p \mathtt{int}\, n : \mathtt{s}}\;\text{app}
}{k : \mathtt{s} \to \mathtt{s}, n : \mathtt{i} \vdash_p k\,(\mathtt{int}\, n) : \mathtt{i} \to \mathtt{s}}\;\text{app}
}{k : \mathtt{s} \to \mathtt{s} \vdash_p \lambda n.\, k\,(\mathtt{int}\, n) : \mathtt{i} \to \mathtt{s}}\;\text{fun}
}{k : \mathtt{s} \to \mathtt{s}; \mathtt{i} \to \mathtt{s} \vdash \lambda n.\, k\,(\mathtt{int}\, n) : \mathtt{i} \to \mathtt{s}; \mathtt{i} \to \mathtt{s}}\;\text{val}
}{\cdot; \color{red}{\mathtt{s}} \vdash \mathcal{S}k.\, \lambda n.\, k\,(\mathtt{int}\, n) : \mathtt{s}; \color{red}{\mathtt{i} \to \mathtt{s}}}\;\text{shift}
}{\cdots}
}{\cdot; \mathtt{s} \vdash \texttt{"t is "}\ \hat{}\ (\mathcal{S}k.\, \lambda n.\, k\,(\mathtt{int}\, n))\ \hat{}\ \texttt{"!"} : \mathtt{s}; \mathtt{i} \to \mathtt{s}}
}{\cdot \vdash_p \langle \texttt{"t is "}\ \hat{}\ (\mathcal{S}k.\, \lambda n.\, k\,(\mathtt{int}\, n))\ \hat{}\ \texttt{"!"} \rangle : \mathtt{i} \to \mathtt{s}}\;\text{reset}
$$

where

$\mathtt{i} \to \mathtt{s}$ abbreviates $\mathtt{i}/\alpha \to \mathtt{s}/\alpha$

$\mathtt{s} \to \mathtt{s}$ abbreviates $\mathtt{s}/\alpha \to \mathtt{s}/\alpha$

for any α.

Example: Type-safe Printf

In the previous slide, (% int) was typed as:

$$\cdot; \mathtt{s} \vdash \mathcal{S}k.\, \lambda n.\, k\,(\mathtt{int}\; n) : \mathtt{s}; \mathtt{i} \to \mathtt{s}$$

What is the type of %, then?

$$\mathtt{let\ \%\ =}\ \lambda \mathit{to\_str}.\, \mathcal{S}k.\, \lambda n.\, k\,(\mathit{to\_str}\; n)$$

We have the following derivation:

$$\dfrac{\mathit{to\_str} : \alpha \to \mathtt{s}; \mathtt{s} \vdash \mathcal{S}k.\, \lambda n.\, k\,(\mathtt{to\_str}\; n) : \mathtt{s}; \alpha \to \mathtt{s}}{\cdot \vdash_p \lambda \mathit{to\_str}.\, \mathcal{S}k.\, \lambda n.\, k\,(\mathtt{to\_str}\; n) : (\alpha \to \mathtt{s})/\mathtt{s} \to \mathtt{s}/(\alpha \to \mathtt{s})}\ \text{fun}$$

For (% int), $\alpha = \mathtt{i}$, whereas for (% str), $\alpha = \mathtt{s}$.

$\Longrightarrow$ We need polymorphism!

Let-Polymorphism

The introduction of let-polymorphism is completely standard...

$$\frac{\Gamma \vdash_p e_1 : \sigma \quad \Gamma, x : \mathsf{Gen}(\sigma; \Gamma); \alpha \vdash e_2 : \tau; \beta}{\Gamma; \alpha \vdash \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 : \tau; \beta} \; \mathsf{let}$$

$$\frac{(x : A \in \Gamma \text{ and } \tau \leq A)}{\Gamma \vdash_p x : \tau} \; \mathsf{var}$$

$\mathsf{Gen}(\sigma; \Gamma)$	:	generalize free type variables in σ that do not occur free in Γ
$\tau \leq A$	:	instanciate generalized type variables to mono types

... except for two places.

(1) answer type polymorphism for delimited continuations

$$\frac{\Gamma, k : \forall t. (\tau/t \rightarrow \alpha/t); \sigma \vdash e : \sigma; \beta}{\Gamma; \alpha \vdash \mathcal{S}k.\, e : \tau; \beta} \; \mathsf{shift}$$

(2) Purity Restriction

Unrestricted polymorphism leads to an unsound type system.

e_1 has to be pure

$$\frac{\boxed{\Gamma \vdash_p e_1 : \sigma} \qquad \Gamma, x : \mathsf{Gen}(\sigma; \Gamma); \alpha \vdash e_2 : \tau; \beta}{\Gamma; \alpha \vdash \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 : \tau; \beta}\ \mathsf{let}$$

- Purity restriction ensures type soundness.
- It is weaker than the value restriction. (Non-value reset expressions can be made polymorphic.)

Technical Results (1)

Theorem. Subject reduction, progress, and unique decomposition, hold. (The type system is sound.)

Thus, a well-typed term does not go wrong. In addition, strong soundness holds: a well-typed term evaluates to the value of the inferred type.

Theorem. The principal type exists. We can infer it using a variant of the algorithm W.

Theorem. The calculus is confluent. The calculus without fix is strongly normalizing.

Note: The calculus with cupto [Gunter et al.'95] is not strongly normalizing.

Theorem. The calculus is compatible with CPS translation.

Correspondence to CPS translation

CPS translation:

$$
\begin{array}{llcl}
\text{(type)} & t^* & = & t \quad \text{for a type variable } t \\
 & (\sigma/\alpha \to \tau/\beta)^* & = & \sigma^* \to (\tau^* \to \alpha^*) \to \beta^* \\
 & (\forall t.\, A)^* & = & \forall t.\, A^* \\
\text{(env)} & (\Gamma, x : A)^* & = & \Gamma^*, x : A^* \\
\text{(val)} & v^* & = & v \\
 & (\lambda x.\, e)^* & = & \lambda x.\, [\![e]\!] \\
\text{(exp)} & [\![v]\!] & = & \lambda \kappa.\, \kappa\, v^* \\
 & [\![e_1\, e_2]\!] & = & \lambda \kappa.\, [\![e_1]\!]\, (\lambda m.\, [\![e_2]\!]\, (\lambda n.\, m\, n\, \kappa)) \\
 & [\![\mathcal{S}k.\, e]\!] & = & \lambda \kappa.\, \textbf{let}\ k = \lambda n \kappa'.\, \kappa'\, (\kappa\, n)\ \textbf{in}\ [\![e]\!]\, (\lambda m.\, m) \\
 & [\![\langle e \rangle]\!] & = & \lambda \kappa.\, \kappa\, ([\![e]\!]\, (\lambda m.\, m)) \\
 & [\![\text{let}\ x = e_1\ \text{in}\ e_2]\!] & = & \lambda \kappa.\, \textbf{let}\ x = [\![e_1]\!]\, (\lambda m.\, m)\ \textbf{in}\ [\![e_2]\!]\, \kappa
\end{array}
$$

- Polymorphism in the source language is expressed by the polymorphism in the target language (a simply-typed λ-calculus with polymorphic let).

Technical Results (2)

Types and Equality are preserved through CPS translation.

Theorem. Preservation of Types:

If $\Gamma; \alpha \vdash e : \tau; \beta$, then $\Gamma^* \vdash [\![e]\!] : (\tau^* \to \alpha^*) \to \beta^*$.
If $\Gamma \vdash_p e : \tau$, then $\Gamma^* \vdash [\![e]\!] : (\tau^* \to \alpha) \to \alpha$ for any α.

Theorem. Preservation of Equality:

If $\Gamma; \alpha \vdash e_1 : \tau; \beta$ and $e_1 \rightsquigarrow^* e_2$, then $[\![e_1]\!] = [\![e_2]\!]$.

Impredicative Polymorphism

The most results carry over to (call-by-value) System F with shift and reset.

(Standard Strategy)

$$
\begin{array}{rcll}
v & ::= & x & \text{variable} \\
 & | & \lambda x : \tau .\, e & \text{abstraction} \\
 & | & \Lambda t .\, e & \text{type abst.} \\
e & ::= & v & \text{value} \\
 & | & e_1\, e_2 & \text{application} \\
 & | & e\,\{\tau\} & \text{type appl.} \\
 & | & \mathcal{S} k : \tau .\, e & \text{shift} \\
 & | & \langle e \rangle & \text{reset}
\end{array}
$$

$$(\Lambda t .\, e)\, \tau \rightsquigarrow e[\tau / t]$$

(ML-like Strategy)

$$
\begin{array}{rcll}
v & ::= & x & \text{variable} \\
 & | & \lambda x : \tau .\, e & \text{abstraction} \\
 & | & \Lambda t .\, v & \text{type abst. 1} \\
e & ::= & v & \text{value} \\
 & | & e_1\, e_2 & \text{application} \\
 & | & e\,\{\tau\} & \text{type appl.} \\
 & | & \mathcal{S} k : \tau .\, e & \text{shift} \\
 & | & \langle e \rangle & \text{reset} \\
 & | & \Lambda t .\, e & \text{type abst. 2}
\end{array}
$$

$$(\Lambda t .\, v)\, \tau \rightsquigarrow v[\tau / t]$$

Difference of the two strategies

Consider: let $f = \langle e \rangle$ in $(f\ f)\ 0$, which is in System F:

$$(\lambda f : \forall t.\, t \to t.\, (f\ \{\mathtt{i} \to \mathtt{i}\})\, (f\ \{\mathtt{i}\})\ 0)\ (\Lambda t.\, \langle e \rangle)$$

- In the standard strategy, evaluation of $\langle e \rangle$ is postponed until $\Lambda t.\, \langle e \rangle$ is applied to a type, giving:

  $$((\Lambda t.\, \langle e \rangle)\ \{\mathtt{i} \to \mathtt{i}\})\ ((\Lambda t.\, \langle e \rangle)\ \{\mathtt{i}\})\ 0$$

  It generalizes "polymorphism by name" [Leroy '93].
- In the ML-like strategy, evaluation of $\langle e \rangle$ is done only once before $\Lambda t.\, \langle e \rangle$ is passed to f.

Technical Results (3)

With the following typing rules (with purity restriction):

$$\frac{\boxed{\Gamma \vdash_p e : \tau}}{\Gamma \vdash_p \Lambda t.\, e : \forall t.\, \tau}\ \text{tabs},\ t \notin \mathrm{FTV}(\Gamma) \qquad \frac{\Gamma; \alpha \vdash e : \forall t.\, \tau; \beta}{\Gamma; \alpha \vdash e\,\{\sigma\} : \tau[\sigma/t]; \beta}\ \text{tapp}$$

we have:

Theorem. The type system is sound for both the strategies.

Furthermore, with a proper definition of CPS translation (omitted; see the paper), we have:

Theorem. Types and equality are preserved through CPS translation for both the strategies.

Related Work

Type systems for control operators:

- Monomorphic type system by Danvy and Filinski [TR '89].
- Harper, Duba, and MacQueen [JFP '93] introduced `callcc` into Standard ML. Strong type soundness does not hold.
- Filinski [POPL '94] presented an implementation of shift/reset in terms of `callcc`. Answer type is fixed.
- Gunter, Rémy, and Riecke [FPCA '95] proposed typed `cupto` operator with strong type soundness. Answer type is fixed.
- Kiselyov, Shan, and Sabry [ICFP '06] introduced shift/reset into OCaml with let-polymorphism.

System F, CPS translation, and control operators:

- Harper and Lillibridge [POPL '93] presented CPS translation from $F\omega$+`callcc` to $F\omega$.

Conclusion

Polymorphism can be introduced naturally, with:

- a type system that mentions answer types,
- purity restriction, and
- answer type polymorphism for captured continuations.

Number of expected results actually hold:

- Strong type soundness, confluence, strong normalization
- Existence of the principal type and type inference algorithm
- Preservation of types and equality through CPS translation

The framework naturally extends to call-by-value System F.

As the foundation of polymorphic delimited continuations is established, we now want deeper understanding and better theories of delimited continuations. E.g., logical relations for polymorphic delimited continuations, relationship to focal parametricity [Hasegawa '06], etc.