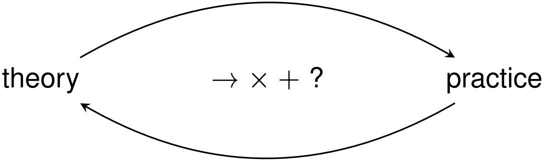
What are these control hierarchies?

Chung-chieh Shan
Rutgers University

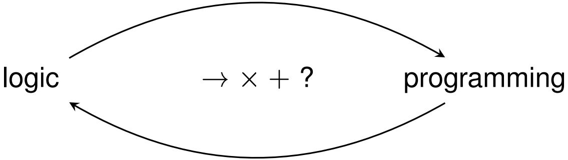
29 May 2011



What are these control hierarchies?

Chung-chieh Shan
Rutgers University

29 May 2011



$$\alpha \to \beta$$

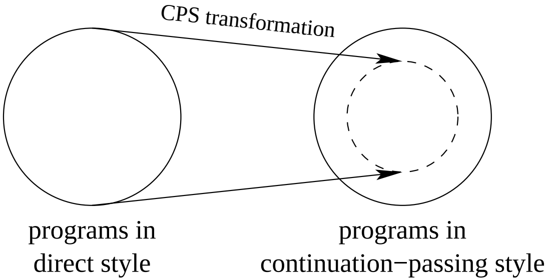
$$\Big\downarrow \text{CPS}$$

$$\alpha \times (\beta \to \omega) \to \omega$$

$\alpha \rightarrow \beta$

$\Big\downarrow$ CPS

$\alpha \times (\beta \rightarrow \omega) \rightarrow \omega$



Danvy & Millikin

$$\alpha \to \beta$$

$$\Big\downarrow \text{CPS}$$

$$\alpha \times (\beta \to \omega) \to \omega$$

Logic guides codifying the pattern: Zeilberger

- Decompose positive vs negative expressions/variables
  function vs context connectives/constructions
- Decompose $\omega_1 \to \beta/\omega_2$ into $(\beta \multimap \omega_1) \triangleright \omega_2$
  $\alpha/\omega_1 \to \beta/\omega_2$ into $((\alpha \to \beta) \multimap \omega_1) \triangleright \omega_2$
- Negatives are polymorphic in the answer type;
  positives are specific in the answer type?

Make nonsense impossible, common sense easy (reflection)?

$$\alpha \to \beta$$

$$\Big\downarrow \text{CPS}$$

$$\alpha \times (\beta \to \omega) \to \omega$$

$$\Big\downarrow \text{CPS}$$

Danvy & Filinski

$$\alpha \times (\beta \times (\omega \to \varpi) \to \varpi) \times (\omega \to \varpi) \to \varpi$$

Make nonsense impossible, common sense easy (reflection)?

What is the total population of the ten largest capitals in the US? Answering these types of complex questions compositionally involves first mapping the questions into logical forms (semantic parsing).

Liang, Jordan & Klein

What is the total population of the ten largest capitals in the US?
Answering these types of complex questions compositionally involves first mapping the questions into logical forms (semantic parsing).

The filtering function F rules out improperly-typed trees . . .
To further reduce the search space . . .

Think of DCS as a higher-level programming language tailored to natural language, which results in programs which are much simpler than the logically-equivalent lambda calculus formulae.

Liang, Jordan & Klein

Alice knows Bob

$$\begin{aligned}\mathtt{Alice} &:: \mathtt{E}\\ \mathtt{Bob} &:: \mathtt{E}\\ \mathtt{know} &:: \mathtt{E} \rightarrow \mathtt{E} \rightarrow \mathtt{Bool}\end{aligned}$$

$$\mathtt{Alice} ⅋ (\mathtt{know}\ \$\ \mathtt{Bob}) :: \mathtt{Bool}$$

Alice knows Bob

$$
\begin{aligned}
&\text{Alice} :: \mathbb{E} \\
&\text{Bob} \quad :: \mathbb{E} \\
&\text{know} :: \mathbb{E} \to \mathbb{E} \to \text{Bool}
\end{aligned}
$$

$$\text{Alice} \;\&\; (\text{know} \;\$\; \text{Bob}) :: \text{Bool}$$

$$
\dfrac{\dfrac{}{\mathbb{E}}\,\text{Alice} \qquad \dfrac{\dfrac{}{\mathbb{E} \to \mathbb{E} \to \text{Bool}}\,\text{know} \qquad \dfrac{}{\mathbb{E}}\,\text{Bob}}{\mathbb{E} \to \text{Bool}}\,\$}{\text{Bool}}\,\&
$$

Alice knows everyone

$$\begin{aligned}
&\mathtt{Alice} :: \mathtt{E}\\
&\mathtt{Bob}\ \ :: \mathtt{E}\\
&\mathtt{know} :: \mathtt{E} \to \mathtt{E} \to \mathtt{Bool}
\end{aligned}$$

$$\mathtt{type}\ \mathbf{M}\,\alpha = (\alpha \to \mathrm{Bool}) \to \mathrm{Bool}$$

$$\begin{aligned}
&\mathtt{everyone} :: \mathbf{M}\,\mathrm{E}\\
&\mathtt{everyone}\ c = \mathrm{all}\ c\ [\mathrm{Alice}, \mathrm{Bob}, ..]
\end{aligned}$$

Barker, de Groote, . . .

Alice knows everyone

$$\begin{aligned}&\mathtt{Alice} :: \mathrm{E}\\&\mathtt{Bob} \;\; :: \mathrm{E}\\&\mathtt{know} :: \mathrm{E} \to \mathrm{E} \to \mathrm{Bool}\end{aligned}$$

$$\mathtt{type}\ \mathbf{M}\,\alpha = (\alpha \to \mathrm{Bool}) \to \mathrm{Bool}$$

$$\begin{aligned}&\mathtt{everyone} :: \mathbf{M}\,\mathrm{E}\\&\mathtt{everyone}\ c = \mathrm{all}\ c\ [\mathrm{Alice}, \mathrm{Bob}, ..]\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{\mathrm{E}}\,\mathtt{Alice}}{\mathbf{M}\,\mathrm{E}}\,\mathtt{return} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{E} \to \mathrm{E} \to \mathrm{Bool}}\,\mathtt{know}}{\mathbf{M}(\mathrm{E} \to \mathrm{E} \to \mathrm{Bool})}\,\mathtt{return} \qquad \dfrac{}{\mathbf{M}\,\mathrm{E}}\,\mathtt{everyone}}{\mathbf{M}(\mathrm{E} \to \mathrm{Bool})}\,\mathrm{liftM2}\ (\$)}{}}{\dfrac{\mathbf{M}\,\mathrm{Bool}}{\mathrm{Bool}}\,(\$\ \mathrm{id})}\,\mathrm{liftM2}\ (\reflectbox{\$})$$

Barker, de Groote, . . .

Alice knows everyone

$$\begin{aligned} \mathtt{Alice} &:: \mathbb{E} \\ \mathtt{Bob} &:: \mathbb{E} \\ \mathtt{know} &:: \mathbb{E} \to \mathbb{E} \to \mathsf{Bool} \end{aligned}$$

$$\mathtt{type}\ \mathbf{M}\,\alpha = (\alpha \to \mathsf{Bool}) \to \mathsf{Bool}$$

$$\begin{aligned} \mathtt{everyone} &:: \mathbf{M}\,\mathbb{E} \\ \mathtt{someone} &:: \mathbf{M}\,\mathbb{E} \\ \mathtt{most} &:: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E} \end{aligned}$$

Someone knows everyone

Alice knows everyone

$$\begin{aligned}&\texttt{Alice} :: \mathbb{E}\\&\texttt{Bob} \;\; :: \mathbb{E}\\&\texttt{know} :: \mathbb{E} \to \mathbb{E} \to \mathsf{Bool}\end{aligned}$$

$$\texttt{type}\ \mathbf{M}\,\alpha = (\alpha \to \mathsf{Bool}) \to \mathsf{Bool}$$

$$\begin{aligned}&\texttt{every} && :: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E}\\&\texttt{some} && :: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E}\\&\texttt{most} && :: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E}\\&\texttt{logician} && :: [\mathbb{E}]\\&\texttt{programmer} && :: [\mathbb{E}]\end{aligned}$$

Someone knows everyone
Most logicians know some programmer

Alice knows everyone

$$\begin{aligned} \mathtt{Alice} &:: \mathbb{E} \\ \mathtt{Bob} &:: \mathbb{E} \\ \mathtt{know} &:: \mathbb{E} \to \mathbb{E} \to \mathrm{Bool} \end{aligned}$$

$$\mathtt{type}\ \mathbf{M}\,\alpha = (\alpha \to \mathrm{Bool}) \to \mathrm{Bool}$$

$$\begin{aligned} \mathtt{every} &:: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E} \\ \mathtt{some} &:: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E} \\ \mathtt{most} &:: [\mathbb{E}] \to \mathbf{M}\,\mathbb{E} \\ \mathtt{logician} &:: [\mathbb{E}] \\ \mathtt{programmer} &:: [\mathbb{E}] \\ \mathtt{from} &:: [\mathbb{E}] \to \mathbb{E} \to [\mathbb{E}] \end{aligned}$$

Someone knows everyone
Most logicians know some programmer
Most logicians know some programmer from Novi Sad

Inverse scope: Someone knows everyone

$$\cfrac{\cfrac{\cfrac{}{\mathbf{M}\,\mathrm{E}}\;\text{someone} \qquad \cfrac{\cfrac{\cfrac{}{\mathrm{E} \to \mathrm{E} \to \mathsf{Bool}}\;\text{know}}{\mathbf{M}(\mathrm{E} \to \mathrm{E} \to \mathsf{Bool})}\;\text{return} \qquad \cfrac{}{\mathbf{M}\,\mathrm{E}}\;\text{everyone}}{\mathbf{M}(\mathrm{E} \to \mathsf{Bool})}\;\text{liftM2}\,(\$)}{\mathbf{M}\,\mathsf{Bool}}\;\text{liftM2}\,(\reflectbox{\$})}{\mathsf{Bool}}\;(\$\,\mathrm{id})$$

Inverse scope: Someone knows everyone

$$\dfrac{\dfrac{\dfrac{}{\mathbf{M}\,\mathrm{E}}\,\text{someone} \qquad \dfrac{\dfrac{\dfrac{}{\mathrm{E} \to \mathrm{E} \to \mathrm{Bool}}\,\text{know}}{\mathbf{M}(\mathrm{E} \to \mathrm{E} \to \mathrm{Bool})}\,\text{return} \qquad \dfrac{}{\mathbf{M}\,\mathrm{E}}\,\text{everyone}}{\mathbf{M}(\mathrm{E} \to \mathrm{Bool})}\,\text{liftM2}\,(\$)}{\mathbf{M}\,\mathrm{Bool}}\,\text{liftM2}\,(\reflectbox{\$})}{\mathrm{Bool}}\,(\$\,\text{id})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{{\color{blue}\mathbf{M}}\mathrm{E}}\,\text{someone}}{{\color{red}\mathbf{M}}({\color{blue}\mathbf{M}}\mathrm{E})}\,\text{return} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{E} \to \mathrm{E} \to \mathrm{Bool}}\,\text{know}}{\mathbf{M}(\mathrm{E} \to \mathrm{E} \to \mathrm{Bool})}\,\text{return}}{\mathbf{M}(\mathbf{M}(\mathrm{E} \to \mathrm{E} \to \mathrm{Bool}))}\,\text{return} \qquad \dfrac{\dfrac{}{{\color{red}\mathbf{M}}\mathrm{E}}\,\text{everyone}}{{\color{red}\mathbf{M}}({\color{blue}\mathbf{M}}\mathrm{E})}\,\text{liftM return}}{{\color{red}\mathbf{M}}({\color{blue}\mathbf{M}}(\mathrm{E} \to \mathrm{Bool}))}\,\text{liftM2}(\text{liftM2}\,(\$))}{{\color{red}\mathbf{M}}({\color{blue}\mathbf{M}}\mathrm{Bool})}\,\text{liftM2}(\text{liftM2}\,(\reflectbox{\$}))}{{\color{red}\mathbf{M}}\mathrm{Bool}}\,\text{liftM}(\$\,\text{id})}{\mathrm{Bool}}\,(\$\,\text{id})$$

Inverse linking: Combining hierarchies?

some programmer from Novi Sad
some programmer from every city May

$$\dfrac{\dfrac{}{[\mathrm{E}] \to \mathbf{M}\,\mathrm{E}}\,\text{some} \qquad \begin{array}{c}\dfrac{\dfrac{}{[\mathrm{E}] \to \mathbf{M}\,\mathrm{E}}\,\text{every} \qquad \dfrac{}{[\mathrm{E}]}\,\text{city}}{\mathbf{M}\,\mathrm{E}}\,\$ \\ \vdots\ \text{programmer from} \\ \mathbf{M}[\mathrm{E}]\end{array}}{\mathbf{M}\,\mathrm{E}}\,???$$

Inverse linking: Combining hierarchies?

some programmer from Novi Sad
some programmer from every city May

$$\dfrac{\dfrac{}{[\mathrm{E}] \to \mathbf{M}\,\mathrm{E}}\,\text{some} \qquad \begin{array}{c} \dfrac{\dfrac{}{[\mathrm{E}] \to \mathbf{M}\,\mathrm{E}}\,\text{every} \quad \dfrac{}{[\mathrm{E}]}\,\text{city}}{\mathbf{M}\,\mathrm{E}}\,\$ \\ \vdots\ \text{programmer from} \\ \mathbf{M}[\mathrm{E}] \end{array}}{\mathbf{M}\,\mathrm{E}}\,???$$

Someone knows everyone
Most logicians know some programmer
Most logicians know some programmer from every city

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\texttt{let}\ t_1 = \cdots\ \texttt{and}\ t_2 = \cdots\ \texttt{and}\ t_3 = \cdots\ \texttt{in}\ \cdots}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\mathtt{let}\ t_1 = \cdots\ \mathtt{and}\ t_2 = \cdots\ \mathtt{and}\ t_3 = \cdots\ \mathtt{in}\ \cdots}$$

$$\begin{aligned}&(\lambda e : \langle\alpha\rangle.\\ &\ \lambda c : \langle\alpha\rangle \to \langle\beta\rangle.\\ &\ \underline{\mathrm{let}\ x =}\ e\ \underline{\mathrm{in}}\ c\underline{x})\\ &\qquad : \ \langle\alpha\rangle \to (\langle\alpha\rangle \to \langle\beta\rangle) \to \langle\beta\rangle\end{aligned}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\texttt{let } t_1 = \cdots \texttt{ and } t_2 = \cdots \texttt{ and } t_3 = \cdots \texttt{ in } \cdots}$$

$$\begin{aligned}&(\lambda e : \langle\alpha\rangle.\\ &\ \lambda c : \langle\alpha\rangle \to \langle\beta\rangle.\\ &\ \underline{\mathrm{let}\ x\ =}\ e\ \underline{\mathrm{in}}\ c\underline{x})\\ &\qquad : \ \langle\alpha\rangle \to (\langle\alpha\rangle \to \langle\beta\rangle) \to \langle\beta\rangle\end{aligned}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\texttt{let } t_1 = \cdots \texttt{ and } t_2 = \cdots \texttt{ and } t_3 = \cdots \texttt{ in } \cdots}$$

$$\begin{aligned}&(\lambda e : \langle\alpha\rangle^{\pi}.\\&\ \lambda c : \langle\alpha\rangle \to \langle\beta\rangle.\\&\ \underline{\text{let } x =}\ e\ \underline{\text{in}}\ c\underline{x})\\&\qquad : \ \langle\alpha\rangle \to (\langle\alpha\rangle \to \langle\beta\rangle) \to \langle\beta\rangle\end{aligned}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\texttt{let } t_1 = \cdots \texttt{ and } t_2 = \cdots \texttt{ and } t_3 = \cdots \texttt{ in } \cdots}$$

$$\begin{aligned}
&(\lambda e : \langle \alpha \rangle^{\pi}. \\
&\ \lambda c : \forall \rho.\ \langle \alpha \rangle^{\pi,\rho} \to \langle \beta \rangle^{\pi,\rho}. \\
&\ \underline{\text{let } x =} \ e \ \underline{\text{in}} \ c\underline{x}) \\
&\qquad : \ \langle \alpha \rangle \to (\langle \alpha \rangle \to \langle \beta \rangle) \to \langle \beta \rangle
\end{aligned}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \leadsto \quad \underline{\mathtt{let}\ t_1 = \cdots\ \mathtt{and}\ t_2 = \cdots\ \mathtt{and}\ t_3 = \cdots\ \mathtt{in}\ \cdots}$$

$$\begin{aligned}&(\lambda e : \langle\alpha\rangle^{\pi,\sigma}.\\ &\ \lambda c : \forall\rho.\ \langle\alpha\rangle^{\pi,\sigma,\rho} \to \langle\beta\rangle^{\pi,\sigma,\rho}.\\ &\ \underline{\mathrm{let}\ x\ =}\ e\ \underline{\mathrm{in}}\ c\underline{x})\\ &\qquad : \forall\sigma.\ \langle\alpha\rangle^{\pi,\sigma} \to (\forall\rho.\ \langle\alpha\rangle^{\pi,\sigma,\rho} \to \langle\beta\rangle^{\pi,\sigma,\rho}) \to \langle\beta\rangle^{\pi,\sigma}\end{aligned}$$

From in-situ quantification to in-situ let-insertion

Write domain-specific code generators in multilevel languages

Nielson & Nielson, Taha

Continuations for code generation, especially let-insertion

Danvy & Filinski, Bondorf, Lawall & Danvy

$$\cdots \underline{+} \cdots \quad \rightsquigarrow \quad \underline{\mathtt{let}\ t_1 = \cdots\ \mathtt{and}\ t_2 = \cdots\ \mathtt{and}\ t_3 = \cdots\ \mathtt{in}\ \cdots}$$

$$\begin{aligned}&(\lambda e : \langle\alpha\rangle^{\pi,\sigma}.\\ &\ \lambda c : \forall\rho.\ \langle\alpha\rangle^{\pi,\sigma,\rho} \to \langle\beta\rangle^{\pi,\sigma,\rho}.\\ &\ \underline{\mathrm{let}\ x =}\ e\ \underline{\mathrm{in}}\ c\underline{x})\\ &\qquad : \forall\sigma.\ \langle\alpha\rangle^{\pi,\sigma} \to (\forall\rho.\ \langle\alpha\rangle^{\pi,\sigma,\rho} \to \langle\beta\rangle^{\pi,\sigma,\rho}) \to \langle\beta\rangle^{\pi,\sigma}\end{aligned}$$

Systematic translation, Kameyama, Kiselyov & Shan
but how does it fit CPS?
Want let-insertion at different scopes.

Summary

Hierarchy 0: composing monad transformers
Hierarchy 1: composing monads (applicative functors)
Hierarchy 2: additional polymorphism at each level

Make nonsense impossible, common sense easy?