A Substructural Type System for Delimited Continuations

Oleg Kiselyov (FNMOC)
Chung-chieh Shan (Rutgers University)

TLCA
June 27, 2007

?

Summary

Small-step abstract interpretation

type systems for ((delimited) control) effects
formalize as a substructural logic

Slogan

Types are abstract expressions (Cousot)
The colon is a turnstile (Lambek)

Outline

- **Delimited continuations**

  Answer types

  Types as abstract interpretation

  Substructural logic

  The $\lambda\xi_0$-calculus with small-step typing

Reset

“#” is the identity continuation (reset [ ]). “$” plugs in a term.

$$\# \, \$ \, \text{“Goldilocks said: ”} \frown (\# \, \$ \, \text{“This porridge is ”} \frown \text{“too hot”} \frown \text{“. ”})$$

$$\rightsquigarrow \# \, \$ \, \text{“Goldilocks said: ”} \frown (\# \, \$ \, \text{“This porridge is ”} \frown \text{“too hot. ”})$$

$$\rightsquigarrow \# \, \$ \, \text{“Goldilocks said: ”} \frown (\# \, \$ \, \text{“This porridge is too hot. ”})$$

$$\rightsquigarrow \# \, \$ \, \text{“Goldilocks said: ”} \frown \text{“This porridge is too hot. ”}$$

$$\rightsquigarrow \# \, \$ \, \text{“Goldilocks said: This porridge is too hot. ”}$$

$$\rightsquigarrow \text{“Goldilocks said: This porridge is too hot. ”}$$

Shift_0

"$\xi_0 k.$" removes and binds k to a continuation.

$$
\begin{aligned}
&\#\,\$\,\text{``Goldilocks said: ''} \frown \\
&\quad (\#\,\$\,\text{``This porridge is ''} \frown \\
&\qquad (\xi_0 k.(k\,\$\,\text{``too hot''}) \frown (k\,\$\,\text{``too cold''}) \frown (k\,\$\,\text{``just right''})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \frown \text{``. ''})
\end{aligned}
$$

$$
\begin{aligned}
\rightsquigarrow\ &\#\,\$\,\text{``Goldilocks said: ''} \frown \\
&\quad ((\#\,\$\,\text{``This porridge is ''} \frown \text{``too hot''} \frown \text{``. ''}) \frown \\
&\quad\ (\#\,\$\,\text{``This porridge is ''} \frown \text{``too cold''} \frown \text{``. ''}) \frown \\
&\quad\ (\#\,\$\,\text{``This porridge is ''} \frown \text{``just right''} \frown \text{``. ''}))
\end{aligned}
$$

$\rightsquigarrow \cdots$

$\rightsquigarrow$ "Goldilocks said:
This porridge is too hot.
This porridge is too cold.
This porridge is just right. "

Applications

Try/catch/reset; throw/abort.

$$\#\,\$\,\text{“Goldilocks said: ”} \frown (\#\,\$\,\text{“Porridge ”} \frown (\xi_0 k.\,\text{“Ouch!”}) \frown \text{“ is my favorite food.”})$$

$$\leadsto \#\,\$\,\text{“Goldilocks said: ”} \frown \text{“Ouch!”}$$

Applications

Try/catch/reset; throw/abort.

$$\#\,\$\,\text{“Goldilocks said: ”} \frown (\#\,\$\,\text{“Porridge ”} \frown (\xi_0 k.\,\text{“Ouch!”}) \frown \text{“ is my favorite food.”})$$
$$\rightsquigarrow \#\,\$\,\text{“Goldilocks said: ”} \frown \text{“Ouch!”}$$

Delimited continuations are useful for:

- backtracking search
- representing monads
- CPS transformation
- partial evaluation
- Web interactions
- mobile code
- linguistics

We need multiple *answer types*, not just string.

Outline

Delimited continuations

- **Answer types**

Types as abstract interpretation

Substructural logic

The $\lambda\xi_0$-calculus with small-step typing

Answer types in the CPS transformation

$$1 < 2$$

$$\lambda k. \quad (\lambda k.\, k 1) \quad (\lambda x. \quad (\lambda k.\, k 2) \quad (\lambda y. \quad k(x < y) \quad))$$

Answer types in the CPS transformation

$$1 < 2$$

$$\overbrace{\lambda k.\ \underbrace{\overbrace{(\lambda k.\,k1)}^{(\text{int}\to T)\to T}\ (\lambda x.\ \underbrace{\overbrace{(\lambda k.\,k2)}^{(\text{int}\to T)\to T}\ (\lambda y.\ \underbrace{k(x<y)}_{T}\ ))}_{T}}_{T}}^{(\text{bool}\to T)\to T}$$

Answer types in the CPS transformation

$$1 < 2$$

$$\overbrace{\lambda k.\ \overbrace{(\lambda k.\,k1)}^{(\mathrm{int}\to T)\to T}\ \underbrace{(\lambda x.\ \overbrace{(\lambda k.\,k2)}^{(\mathrm{int}\to T)\to T}\ \underbrace{(\lambda y.\ \underbrace{k(x<y)}_{T}\ ))}_{T}}_{T}}^{(\mathrm{bool}\to T)\to T}$$

Answer types in the CPS transformation

$$1 < 2$$

$$(\xi_0 k.\,\text{"Ouch!"}) < 2$$

$$\overbrace{\lambda k.\ \underbrace{\overbrace{(\lambda k.\,\text{"Ouch!"})}^{(\text{int}\to T)\to\text{string}}\ (\lambda x.\ \underbrace{\overbrace{(\lambda k.\,k2)}^{(\text{int}\to T)\to T}\ \ (\lambda y.\ \underbrace{k(x<y)}_{T}\ ))}_{T}}_{\text{string}}}^{(\text{bool}\to T)\to\text{string}}$$

Answer types in the CPS transformation

$$
\begin{aligned}
1 &< 2 \\
(\xi_0 k.\,\text{“Ouch!”}) &< 2 \\
1 &< (\xi_0 k.\,\text{‘c’})
\end{aligned}
$$

$$
\overbrace{\lambda k.\ \underbrace{\overbrace{(\lambda k.\,k1)}^{(\text{int}\to\text{char})\to\text{char}}\ \ (\lambda x.\ \underbrace{\overbrace{(\lambda k.\,\text{‘c’})}^{(\text{int}\to T)\to\text{char}}\ \ (\lambda y.\ \underbrace{k(x<y)}_{T}\ ))}_{\text{char}}}_{\text{char}}}^{(\text{bool}\to T)\to\text{char}}
$$

Answer types in the CPS transformation

$$
\begin{aligned}
1 &< 2 \\
(\xi_0 k.\,\text{“Ouch!”}) &< 2 \\
1 &< (\xi_0 k.\,\text{‘c’}) \\
\color{red}{(\xi_0 k.\,\text{“Ouch!”})} &< (\xi_0 k.\,\text{‘c’})
\end{aligned}
$$

$$
\overbrace{\lambda k.\ \underbrace{\overbrace{(\lambda k.\,\color{red}{\text{“Ouch!”}})}^{(\text{int}\to\color{red}{\text{char}})\to\color{red}{\text{string}}}\ (\lambda x.\ \underbrace{\overbrace{(\lambda k.\,\text{‘c’})}^{(\text{int}\to T)\to\color{red}{\text{char}}}\ (\lambda y.\ \underbrace{k(x<y)}_{T})}_{\color{red}{\text{char}}}))}_{\color{red}{\text{string}}}}^{(\text{bool}\to T)\to\color{red}{\text{string}}}
$$

Evaluation order chains together *initial* and *final* answer types.

Desideratum 1: Changing the answer type

Need to change the answer type to:

- create functions
- find list prefixes
- represent *parameterized monads*
- analyze questions and polarity in natural language
- Ghani & Johann, “generalized continuations” yesterday
  newtype Ran $g\ h\ a = \mathrm{Ran}\ (\forall b.\,(a \to g\,b) \to h\,b)$

Desideratum 1: Changing the answer type

Need to change the answer type to:

- create functions
- find list prefixes
- represent *parameterized monads*
- analyze questions and polarity in natural language
- Ghani & Johann, "generalized continuations" yesterday
  newtype Ran $g\ h\ a = \text{Ran}\ (\forall b. (a \to g\, b) \to h\, b)$

For example (Danvy & Filinski):

$$\frac{[x \mapsto \sigma]\rho, \alpha \vdash E : \tau, \beta}{\rho, \delta \vdash \lambda x. E : (\sigma/\alpha \to \tau/\beta), \delta.}$$

The 4-ary type constructor $/\!\to\!/$ fits exactly two answer types.

Desideratum 1: Changing the answer type

Need to change the answer type to:

- create functions
- find list prefixes
- represent *parameterized monads*
- analyze questions and polarity in natural language
- Ghani & Johann, "generalized continuations" yesterday
  newtype Ran $g\ h\ a = \mathrm{Ran}\ (\forall b.\,(a \to g\,b) \to h\,b)$

For example (Danvy & Filinski):

$$\frac{[x \mapsto \sigma]\rho, \overbrace{\alpha \vdash E : \tau, \beta}^{(\tau \to \alpha) \to \beta}}{\rho, \delta \vdash \lambda x.E : (\underbrace{\sigma/\alpha \to \tau/\beta}_{\sigma \to (\tau \to \alpha) \to \beta}), \delta.}$$

The 4-ary type constructor $/\!\to\!/$ fits exactly two answer types.

Desideratum 2: Reaching beyond the nearest delimiter

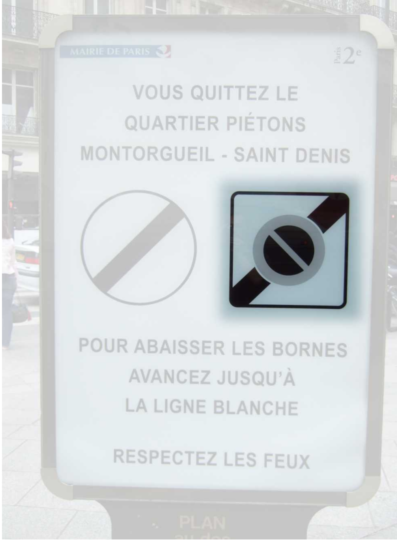
Need to reach beyond the nearest delimiter to:

- combine multiple monadic effects
- normalize λ-terms with sums
- simulate exceptions and mutable references
- simulate dynamic binding
- encode process calculi?

Lacuna

Unfortunately—

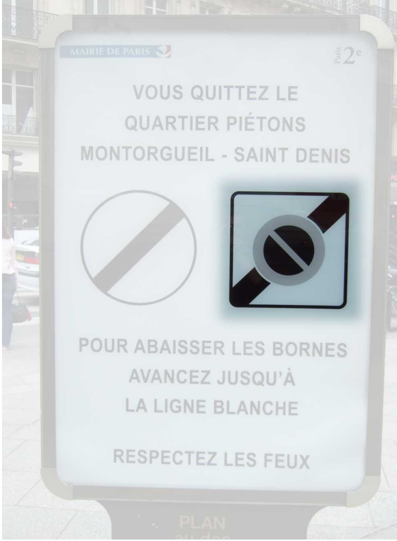
- No existing type system subsumes all others.
- Answer types (effect annotations) on judgments and arrows obscure logical interpretation.



Lacuna

Unfortunately—

- No existing type system subsumes all others.
- Answer types (effect annotations) on judgments and arrows obscure logical interpretation.



Our new system allows an arbitrary number of changing answer types, using small-step abstract interpretation and only binary connectives.

Lacuna

$$\xi_0 k.\text{“Ouch!”} \quad : \quad (\text{int} \to T) \to \text{string}$$

Our new system allows an arbitrary number of changing answer types, using small-step abstract interpretation and only binary connectives.

Lacuna

$$\xi_0 k.\text{“Ouch!”} \quad : \quad (\text{int} \uparrow T) \downarrow \text{string}$$

Our new system allows an arbitrary number of changing answer types, using small-step abstract interpretation and only binary connectives.

Lacuna

$$
\begin{array}{rcl}
\xi_0 k.\,\text{“Ouch!”} & : & (\mathsf{int} \uparrow T) \downarrow \mathsf{string} \\
\\
1 < 2 & : & \mathsf{bool} \\
1 < 2 & : & (\mathsf{bool} \uparrow T) \downarrow T \\
(\xi_0 k.\,\text{“Ouch!”}) < 2 & : & (\mathsf{bool} \uparrow T) \downarrow \mathsf{string} \\
(\xi_0 k.\,\text{“Ouch!”}) < (\xi_0 k.\,\text{‘c’}) & : & (\mathsf{bool} \uparrow T) \downarrow \mathsf{string}
\end{array}
$$

Our new system allows an arbitrary number of changing answer types, using small-step abstract interpretation and only binary connectives.

Outline

Delimited continuations

Answer types

► **Types as abstract interpretation**

Substructural logic

The $\lambda\xi_0$-calculus with small-step typing

Big step

Operational semantics

$$\dfrac{\dfrac{\dfrac{}{1 \Downarrow 1} \quad \dfrac{}{2 \Downarrow 2}}{1 + 2 \Downarrow 3} \quad \dfrac{\dfrac{}{3 \Downarrow 3} \quad \dfrac{}{4 \Downarrow 4}}{3 + 4 \Downarrow 7}}{1 + 2 < 3 + 4 \Downarrow \mathit{true}}$$

Abstract interpretation

$$\dfrac{\dfrac{\dfrac{}{1 : \mathrm{int}} \quad \dfrac{}{2 : \mathrm{int}}}{1 + 2 : \mathrm{int}} \quad \dfrac{\dfrac{}{3 : \mathrm{int}} \quad \dfrac{}{4 : \mathrm{int}}}{3 + 4 : \mathrm{int}}}{1 + 2 < 3 + 4 : \mathrm{bool}}$$

For control effects, need small steps.

Small step

Operational semantics

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{true} \Downarrow \mathit{true}}}{3 < 7 \Downarrow \mathit{true}}}{3 < 3 + 4 \Downarrow \mathit{true}}}{1 + 2 < 3 + 4 \Downarrow \mathit{true}} \qquad \uparrow \text{time}$$

Abstract interpretation

$$\dfrac{\dfrac{\dfrac{[x : \text{int}]^1 \quad [y : \text{int}]^2 \quad [z : \text{bool}]^3}{x < y : \text{bool}}\,3}{x < 3 + 4 : \text{bool}}\,2}{1 + 2 < 3 + 4 : \text{bool}}\,1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\text{bool} : \text{bool}}}{\text{int} < \text{int} : \text{bool}}}{\text{int} < 3 + 4 : \text{bool}}}{1 + 2 < 3 + 4 : \text{bool}}$$

An abstract value is a linear hypothesis. The colon is a turnstile.

Abstract expressions and abstract evaluation contexts

Plugging works in both directions.

$$\begin{aligned} (U \uparrow T) \,\$\, U \quad &: \quad T \\ S \,\$\, (S \downarrow T) \quad &: \quad T \end{aligned}$$

Also need hypothetical reasoning.

Abstract expressions and abstract evaluation contexts

$$\frac{}{1 < 2 \qquad : \ \text{bool}}$$

Abstract expressions and abstract evaluation contexts

$$\cfrac{\cfrac{\cfrac{[k : \mathrm{bool} \uparrow T]^1 \quad [z : \mathrm{bool}]^2}{k \,\$\, z : T}}{k \,\$\, (1 < 2) : T}\,2}{1 < 2 : (\mathrm{bool} \uparrow T) \downarrow T}\,1$$

$$\cfrac{\cfrac{}{(\mathrm{bool} \uparrow T) \,\$\, \mathrm{bool} : T}}{(\mathrm{bool} \uparrow T) \,\$\, (\;1 < 2\;) : T}$$

$$\cfrac{}{1 < 2 \qquad : (\mathrm{bool} \uparrow T) \downarrow T}$$

Duality between $\uparrow$ and $\downarrow$; bool is a subtype of $(\mathrm{bool} \uparrow T) \downarrow T$.

Abstract expressions and abstract evaluation contexts

$$\frac{\dfrac{\dfrac{}{\text{“Ouch!”} : \text{string}}}{\xi_0 k.\text{“Ouch!”} : (\text{int} \uparrow T) \downarrow \text{string}} \qquad \dfrac{\dfrac{}{(\text{bool} \uparrow T)\,\$\,\text{bool} : T}}{(\text{bool} \uparrow T)\,\$\,(\text{int} < 2) : T}}{\dfrac{(\text{bool} \uparrow T)\,\$\,((\xi_0 k.\text{“Ouch!”}) < 2) : \text{string}}{(\xi_0 k.\text{“Ouch!”}) < 2 : (\text{bool} \uparrow T) \downarrow \text{string}}}$$

Typing derivation is CPS transformation; ξ_0 is abstraction like λ.

Abstract expressions and abstract evaluation contexts

$$
\cfrac{
  \cfrac{\cfrac{}{\text{“Ouch!”} : \mathsf{string}}}{\xi_0 k.\,\text{“Ouch!”} : (\mathsf{int} \uparrow \mathsf{char}) \downarrow \mathsf{string}}
  \qquad
  \cfrac{
    \cfrac{\cfrac{}{\text{‘c’} : \mathsf{char}}}{\xi_0 k.\,\text{‘c’} : (\mathsf{int} \uparrow T) \downarrow \mathsf{char}}
    \quad
    \cfrac{\cfrac{}{(\mathsf{bool} \uparrow T) \mathbin{\$} \mathsf{bool} : T}}{(\mathsf{bool} \uparrow T) \mathbin{\$} (\mathsf{int} < \mathsf{int}) : T}
  }{(\mathsf{bool} \uparrow T) \mathbin{\$} \big(\mathsf{int} < (\xi_0 k.\,\text{‘c’})\big) : \mathsf{char}}
}{
  \cfrac{(\mathsf{bool} \uparrow T) \mathbin{\$} \big((\xi_0 k.\,\text{“Ouch!”}) < (\xi_0 k.\,\text{‘c’})\big) : \mathsf{string}}{(\xi_0 k.\,\text{“Ouch!”}) < (\xi_0 k.\,\text{‘c’}) : (\mathsf{bool} \uparrow T) \downarrow \mathsf{string}}
}
$$

time ↑

The rest is details: to abstractly decompose an expression into a redex and an evaluation context, use *substructural logic*.

Outline

Delimited continuations

Answer types

Types as abstract interpretation

► **Substructural logic**

The $\lambda\xi_0$-calculus with small-step typing

Directionality controls combination

An expression is a structure.
Juxtaposition is tensor, but neither commutative nor associative.

Natural deduction: T, T' are types, E, F are structures of types.

$$\frac{}{T : T} \quad \frac{AF : T'}{F : T \backslash T'} \backslash\mathrm{I} \quad \frac{E : T \quad F : T \backslash T'}{EF : T'} \backslash\mathrm{E} \quad \frac{FA : T'}{F : T'/T} /\mathrm{I} \quad \frac{F : T'/T \quad E : T}{FE : T'} /\mathrm{E}$$

$$\frac{\text{Alice} : \text{person} \qquad \dfrac{\text{saw} : (\text{person} \backslash \text{bool})/\text{person} \quad \text{Bob} : \text{person}}{\text{saw Bob} : \text{person} \backslash \text{bool}} /\mathrm{E}}{\text{Alice (saw Bob)} : \text{bool}} \backslash\mathrm{E}$$

$T \to T'$ is T'/T.

Modes construct syntax

The type of "woman" cannot be person\bool or bool/person.

$$\frac{A n F : T'}{F : T \backslash_n T'} \backslash_n\mathrm{I} \quad \frac{E : T \quad F : T \backslash_n T'}{E n F : T'} \backslash_n\mathrm{E} \quad \frac{F n A : T'}{F : T' /_n T} /_n\mathrm{I} \quad \frac{F : T' /_n T \quad E : T}{F n E : T'} /_n\mathrm{E}$$

The new binary mode n is *internal* (unspeakable).

$$\frac{\text{Alice} : \text{person} \quad \text{woman} : \text{person} \backslash_n \text{bool}}{\text{Alice } n \text{ woman} : \text{bool}} \backslash_n\mathrm{E}$$

Modes interact through stipulated *structural rules*.

Modes construct syntax

A binary mode without products

$$\frac{AnF : T'}{F : T\backslash_n T'}\ \backslash_n\mathrm{I} \qquad \frac{E : T \quad F : T\backslash_n T'}{EnF : T'}\ \backslash_n\mathrm{E} \qquad \frac{FnA : T'}{F : T'/_n T}\ /_n\mathrm{I} \qquad \frac{F : T'/_n T \quad E : T}{FnE : T'}\ /_n\mathrm{E}$$

A unary mode with products

$$\frac{\langle E\rangle : T}{E : \Box^{\downarrow}T}\ \Box^{\downarrow}\mathrm{I} \qquad \frac{E : \Box^{\downarrow}T}{\langle E\rangle : T}\ \Box^{\downarrow}\mathrm{E} \qquad \frac{E : T}{\langle E\rangle : \Diamond T}\ \Diamond\mathrm{I} \qquad \frac{E : \Diamond T \quad \Gamma[\langle T\rangle] : T'}{\Gamma[E] : T'}\ \Diamond\mathrm{E}$$

For delimited continuations, reify evaluation contexts as coterms.

We add two binary modes for coterms:	,
	;
a nullary mode for coterms:	#
a binary mode for plugging:	$
a unary mode for values:	(implicit)

Modes construct syntax

A binary mode without products

$$\frac{AnF : T'}{F : T\backslash_n T'}\ \backslash_n\mathrm{I} \quad \frac{E : T \quad F : T\backslash_n T'}{EnF : T'}\ \backslash_n\mathrm{E} \quad \frac{FnA : T'}{F : T'/_n T}\ /_n\mathrm{I} \quad \frac{F : T'/_n T \quad E : T}{FnE : T'}\ /_n\mathrm{E}$$

A unary mode with products

$$\frac{\langle E\rangle : T}{E : \Box^{\downarrow} T}\ \Box^{\downarrow}\mathrm{I} \quad \frac{E : \Box^{\downarrow} T}{\langle E\rangle : T}\ \Box^{\downarrow}\mathrm{E} \quad \frac{E : T}{\langle E\rangle : \Diamond T}\ \Diamond\mathrm{I} \quad \frac{E : \Diamond T \quad \Gamma[\langle T\rangle] : T'}{\Gamma[E] : T'}\ \Diamond\mathrm{E}$$

For delimited continuations, reify evaluation contexts as coterms.

We add two binary modes for coterms:	E, C	$C\big[[\ ]E\big]$
	$C; V$	$C\big[V[\ ]\big]$
a nullary mode for coterms:	$\#$	reset [ ]
a binary mode for plugging:	$C \$ E$	$C[E]$
a unary mode for values:	V	

Structural rules express evaluation order

$$C\,\$\,FE = E, C\,\$\,F \qquad C\,\$\,VE = C; V\,\$\,E \qquad V = \#\,\$\,V$$

$$\frac{\Gamma[C\,\$\,FE] : T}{\Gamma[E, C\,\$\,F] : T} \qquad \frac{\Gamma[C\,\$\,VE] : T}{\Gamma[C; V\,\$\,E] : T} \qquad \frac{\Gamma[V] : T}{\Gamma[\#\,\$\,V] : T}$$

$$\begin{aligned} \#\,\$\,\big(V_1(V_2V_3)\big)V_4 &= (V_4, \#)\,\$\,V_1(V_2V_3) \\ &= \big(V_2V_3, (V_4, \#)\big)\,\$\,V_1 \\ &= \big((V_4, \#); V_1\big)\,\$\,V_2V_3 \end{aligned}$$

Our coterm type $T \uparrow T'$ is $T' /_{\$} T$.

Our impure term type $T \downarrow T'$ is $T \backslash_{\$} T'$.

Outline

Delimited continuations

Answer types

Types as abstract interpretation

Substructural logic

► **The $\lambda\xi_0$-calculus with small-step typing**

Terms	$E, F ::= V \mid FE \mid C \mathbin{\$} E \mid \xi_0 k.\, E$
Values	$V ::= x \mid \lambda x.\, E$
Coterms	$C ::= k \mid \# \mid E, C \mid C; V$
Types	$T ::= U \mid S \downarrow T$
Pure types	$U ::= U \to T \mid \text{string} \mid \text{int} \mid \cdots$
Cotypes	$S ::= U \uparrow T$
Transitions	

$$C_1 \mathbin{\$} \cdots \mathbin{\$} C_n \mathbin{\$} (\lambda x.\, E)V \rightsquigarrow C_1 \mathbin{\$} \cdots \mathbin{\$} C_n \mathbin{\$} E\{x \mapsto V\}$$

$$C_1 \mathbin{\$} \cdots \mathbin{\$} C_n \mathbin{\$} C \mathbin{\$} (\xi_0 k.\, E) \rightsquigarrow C_1 \mathbin{\$} \cdots \mathbin{\$} C_n \mathbin{\$} E\{k \mapsto C\}$$

Reset: dynamic semantics

Alternate between refocusing and reducing.

$$
\begin{aligned}
& \#\,\$\,\text{“Goldilocks said: ”} \frown \\
& \quad (\#\,\$\,\text{“This porridge is ”} \frown \text{“too hot”} \frown \text{“. ”}) \\
= {} & \#;(\text{“Goldilocks said: ”}\frown)\,\$ \\
& \quad (\#;(\text{“This porridge is ”}\frown)\,\$\,\text{“too hot”} \frown \text{“. ”}) \\
\rightsquigarrow {} & \#;(\text{“Goldilocks said: ”}\frown)\,\$ \\
& \quad (\#;(\text{“This porridge is ”}\frown)\,\$\,\text{“too hot. ”}) \\
= {} & \#;(\text{“Goldilocks said: ”}\frown)\,\$\,(\#\,\$\,\text{“This porridge is ”} \frown \text{“too hot. ”}) \\
\rightsquigarrow {} & \#;(\text{“Goldilocks said: ”}\frown)\,\$\,(\#\,\$\,\text{“This porridge is too hot. ”}) \\
= {} & \#\,\$\,\text{“Goldilocks said: ”} \frown \text{“This porridge is too hot. ”} \\
\rightsquigarrow {} & \#\,\$\,\text{“Goldilocks said: This porridge is too hot. ”} \\
= {} & \text{“Goldilocks said: This porridge is too hot. ”}
\end{aligned}
$$

Shift$_0$: dynamic semantics

$$
\begin{aligned}
&\#\,\$\,\text{“Goldilocks said: ”}\frown \\
&\quad(\#\,\$\,\text{“This porridge is ”}\frown \\
&\qquad(\xi_0 k.(k\,\$\,\text{“too hot”})\frown(k\,\$\,\text{“too cold”})\frown(k\,\$\,\text{“just right”})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\frown\text{“. ”})
\end{aligned}
$$

$$
\begin{aligned}
= {}&\#;(\text{“Goldilocks said: ”}\frown)\,\$ \\
&\quad((\text{“. ”},(\#;(\text{“This porridge is ”}\frown)));\frown)\,\$ \\
&\qquad(\xi_0 k.(k\,\$\,\text{“too hot”})\frown(k\,\$\,\text{“too cold”})\frown(k\,\$\,\text{“just right”}))
\end{aligned}
$$

$$
\begin{aligned}
\rightsquigarrow {}&\#;(\text{“Goldilocks said: ”}\frown)\,\$ \\
&\quad((((\text{“. ”},(\#;(\text{“This porridge is ”}\frown)));\frown)\,\$\,\text{“too hot”})\frown \\
&\quad\;(((\text{“. ”},(\#;(\text{“This porridge is ”}\frown)));\frown)\,\$\,\text{“too cold”})\frown \\
&\quad\;(((\text{“. ”},(\#;(\text{“This porridge is ”}\frown)));\frown)\,\$\,\text{“just right”}))
\end{aligned}
$$

$$
\begin{aligned}
= {}&\#;(\text{“Goldilocks said: ”}\frown)\,\$ \\
&\quad((\#\,\$\,\text{“This porridge is ”}\frown\text{“too hot”}\frown\text{“. ”})\frown \\
&\quad\;(\#\,\$\,\text{“This porridge is ”}\frown\text{“too cold”}\frown\text{“. ”})\frown \\
&\quad\;(\#\,\$\,\text{“This porridge is ”}\frown\text{“just right”}\frown\text{“. ”}))
\end{aligned}
$$

$$\rightsquigarrow \cdots$$

Terms	$E, F ::= V \mid FE \mid C \,\$\, E \mid \xi_0 k.E$
Values	$V ::= x \mid \lambda x.E$
Coterms	$C ::= k \mid \# \mid E, C \mid C; V$
Types	$T ::= U \mid S \downarrow T$
Pure types	$U ::= U \rightarrow T \mid \mathrm{string} \mid \mathrm{int} \mid \cdots$
Cotypes	$S ::= U \uparrow T$

$$\frac{\begin{matrix}[x:U]\\ \vdots \\ E:T\end{matrix}}{\lambda x.E : U \rightarrow T}\,\lambda \qquad \frac{\begin{matrix}[k:S]\\ \vdots \\ E:T\end{matrix}}{\xi_0 k.E : S \downarrow T}\,\xi_0 \qquad \frac{F:U \quad \begin{matrix}[x:U]\\ \vdots \\ E[x]:T\end{matrix}}{E[F]:T}\,E[U]$$

$$\frac{\begin{matrix}[x:U]\\ \vdots \\ Vx:T\end{matrix}}{V : U \rightarrow T}\rightarrow\mathrm{I} \qquad \frac{\begin{matrix}[k:S]\\ \vdots \\ k\,\$\,E:T\end{matrix}}{E : S \downarrow T}\downarrow\mathrm{I} \qquad \frac{\begin{matrix}[x:U]\\ \vdots \\ C\,\$\,x:T\end{matrix}}{C : U \uparrow T}\uparrow\mathrm{I}$$

$$\frac{F : U \rightarrow T \quad E : U}{FE : T}\rightarrow\mathrm{E} \qquad \frac{C : S \quad E : S \downarrow T}{C\,\$\,E : T}\downarrow\mathrm{E} \qquad \frac{C : U \uparrow T \quad E : U}{C\,\$\,E : T}\uparrow\mathrm{E}$$

Reset: static semantics

Compute cotype by hypothetical reasoning.

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}[x:\text{string}]\\ \vdots\\ \#\;\$\;(\text{“Tpi”}\frown x\frown\text{“. ”}):\text{string}\end{array}}{((\text{“. ”},(\#;(\text{“Tpi”}\frown)));\frown)\;\$\;x:\text{string}}=}{((\text{“. ”},(\#;(\text{“Tpi”}\frown)));\frown):\text{string}\uparrow\text{string}}\uparrow\mathrm{I}\qquad\dfrac{}{\text{“too hot”}:\text{string}}}{((\text{“. ”},(\#;(\text{“Tpi”}\frown)));\frown)\;\$\;\text{“too hot”}:\text{string}}\uparrow\mathrm{E}}{\begin{array}{c}\#\;\$\;(\text{“Tpi”}\frown\text{“too hot”}\frown\text{“. ”}):\text{string}\\ \vdots\end{array}}=$$

Shift$_0$: static semantics

Compute cotype by hypothetical reasoning.

$$
\dfrac{
  \dfrac{
    \dfrac{
      \begin{array}{c}[x : \text{string}] \\ \vdots \end{array} \\
      \#\ \$\ (\text{“Tpi”} \frown x \frown \text{“. ”}) : \text{string}
    }{
      ((\text{“. ”}, (\#; (\text{“Tpi”}\frown))); \frown)\ \$\ x : \text{string}
    } =
  }{
    ((\text{“. ”}, (\#; (\text{“Tpi”}\frown))); \frown) : \color{red}{\text{string} \uparrow \text{string}}
  } \uparrow\text{I}
  \quad
  \begin{array}{c} \vdots \\ \xi_0 k. (k\ \$\ \text{“too hot”}) \frown \cdots \\ : \color{red}{(\text{string} \uparrow \text{string}) \downarrow \text{string}} \end{array}
}{
  \dfrac{
    ((\text{“. ”}, (\#; (\text{“Tpi”}\frown))); \frown)\ \$\ \text{“too hot”} : \text{string}
  }{
    \begin{array}{c} \#\ \$\ (\text{“Tpi”} \frown \text{“too hot”} \frown \text{“. ”}) : \text{string} \\ \vdots \end{array}
  } =
} \color{red}{\downarrow\text{E}}
$$

Type checking algorithm

$$length(\xi_0 k : \mathrm{string} \uparrow \mathrm{bool}.\, \text{‘x’})$$

Say that *length* is a string $\rightarrow$ int function.

Type checking algorithm

Implemented in Twelf.

$$
\dfrac{
\dfrac{
\dfrac{[k : \mathrm{string} \uparrow \mathrm{bool}]^{1} \\ \vdots}{\text{‘x’} : \mathrm{char}}
\quad
}{
(\xi_0 k : \mathrm{string} \uparrow \mathrm{bool}.\ \text{‘x’}) \Rightarrow (\mathrm{string} \uparrow \mathrm{bool}) \downarrow \mathrm{char}
}\,1
\qquad
\dfrac{
\dfrac{
\dfrac{[x : \mathrm{string}]^{2}}{\mathit{length}\, x \Rightarrow \mathrm{int}}
\quad
\dfrac{[n : \mathrm{int}]^{3}}{\mathrm{int} \uparrow \mathrm{bool} \Leftarrow \langle\rangle \,¢\, n : \mathrm{bool}}
}{
\mathrm{int} \uparrow \mathrm{bool} \Leftarrow \langle\rangle \,¢\, \mathit{length}\, x : \mathrm{bool}
}\,3
}{
\mathrm{int} \uparrow \mathrm{bool} \Leftarrow \langle\rangle ; \mathit{length} \,¢\, x : \mathrm{bool}
}
}{
\dfrac{
\langle\rangle ; \mathit{length} \,¢\, (\xi_0 k : \mathrm{string} \uparrow \mathrm{bool}.\ \text{‘x’}) \Rightarrow (\mathrm{int} \uparrow \mathrm{bool}) \downarrow \mathrm{char}
}{
\langle\rangle \,¢\, \mathit{length}(\xi_0 k : \mathrm{string} \uparrow \mathrm{bool}.\ \text{‘x’}) \Rightarrow (\mathrm{int} \uparrow \mathrm{bool}) \downarrow \mathrm{char}
}
}\,2
$$

Tridirectional type-checking?

Summary

Small-step abstract interpretation

type systems for ((delimited) control) effects
formalize as a substructural logic

Slogan

Types are abstract expressions (Cousot)
The colon is a turnstile (Lambek)

Discoveries

Binding and evaluation contexts are related, but the latter is linear.
The typing derivation of a direct-style program desugars it into continuation-passing style.

Code online

`http://pobox.com/~oleg/ftp/packages/small-step-typechecking.tar.gz`