

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.

\$ “Goldilocks said: ” ^

(# \$ “This porridge is ” ^ “too hot” ^ “. ”)

~> # \$ “Goldilocks said: ” ^ (# \$ “This porridge is ” ^ “too hot. ”)

~> # \$ “Goldilocks said: ” ^ (# \$ “This porridge is too hot. ”)

~> # \$ “Goldilocks said: ” ^ “This porridge is too hot. ”

~> # \$ “Goldilocks said: This porridge is too hot. ”

~> “Goldilocks said: This porridge is too hot. ”

Shift₀

“ $\xi_0 k$.” removes and binds k to a continuation.

\$ “Goldilocks said: ” \wedge
 (# \$ “This porridge is ” \wedge
 ($\xi_0 k$.(k \$ “too hot”) \wedge (k \$ “too cold”) \wedge (k \$ “just right”))
 \wedge “. ”)

\rightsquigarrow # \$ “Goldilocks said: ” \wedge
 ((# \$ “This porridge is ” \wedge “too hot” \wedge “. ”) \wedge
 (# \$ “This porridge is ” \wedge “too cold” \wedge “. ”) \wedge
 (# \$ “This porridge is ” \wedge “just right” \wedge “. ”))

\rightsquigarrow ...

\rightsquigarrow “Goldilocks said:
 This porridge is too hot.
 This porridge is too cold.
 This porridge is just right. ”

Applications

Try/catch/reset; **throw/abort**.

\$ "Goldilocks said: " ^

(# \$ "Porridge " ^ (*ξ₀k*. "Ouch!") ^ " is my favorite food.")

↪ # \$ "Goldilocks said: " ^ "Ouch!"

Applications

Try/catch/reset; **throw/abort**.

\$ "Goldilocks said: " ^

(# \$ "Porridge " ^ (ξ_0 k. "Ouch!") ^ " is my favorite food.")

↪ # \$ "Goldilocks said: " ^ "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

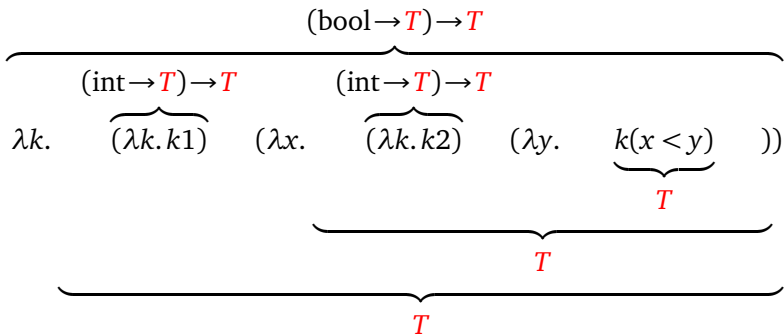
Answer types in the CPS transformation

1 < 2

$\lambda k. (\lambda k. k1) (\lambda x. (\lambda k. k2) (\lambda y. k(x < y)))$

Answer types in the CPS transformation

1 < 2



Answer types in the CPS transformation

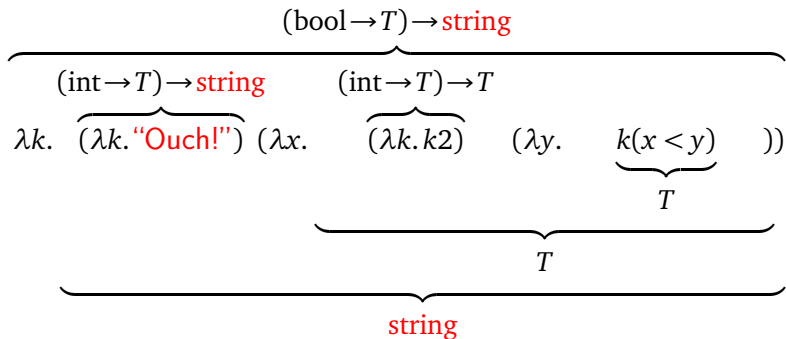
1 < 2

$$\begin{array}{c} \overbrace{\lambda k. \quad \underbrace{(\lambda x. \quad \underbrace{(\lambda y. \quad \underbrace{k(x < y)}_T) \quad)}}_T \quad)}_{(int \rightarrow T) \rightarrow T} \quad \underbrace{(\lambda k. k1)}_{(int \rightarrow T) \rightarrow T} \quad)}_{(bool \rightarrow T) \rightarrow T} \end{array}$$

Answer types in the CPS transformation

$1 < 2$

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

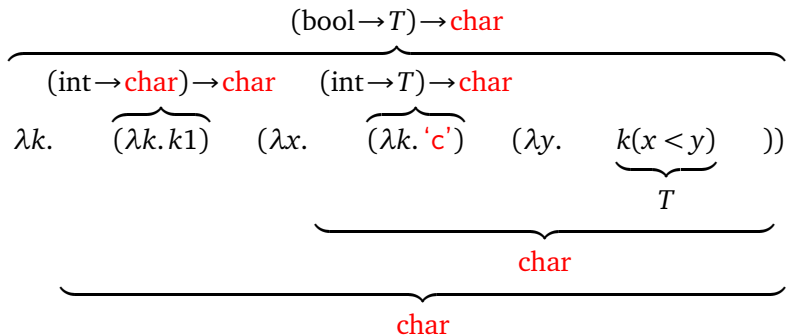


Answer types in the CPS transformation

$1 < 2$

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

$1 < (\xi_0 k. 'c')$



Answer types in the CPS transformation

$$\begin{aligned} & 1 < 2 \\ & (\xi_0 k. \text{"Ouch!"}) < 2 \\ & 1 < (\xi_0 k. 'c') \\ & (\xi_0 k. \text{"Ouch!"}) < (\xi_0 k. 'c') \end{aligned}$$

$$\begin{array}{c} \text{(bool} \rightarrow T) \rightarrow \text{string} \\ \hline \text{(int} \rightarrow \text{char}) \rightarrow \text{string} \quad \text{(int} \rightarrow T) \rightarrow \text{char} \\ \lambda k. \underbrace{(\lambda k. \text{"Ouch!"})}_{\text{char}} \quad (\lambda x. \underbrace{(\lambda k. 'c')}_{\text{char}} \quad (\lambda y. \underbrace{k(x < y)}_T) \quad)) \\ \hline \text{char} \\ \hline \text{string} \end{array}$$

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 $\text{Ran } g \ h \ a = \text{Ran } (\forall b. (a \rightarrow g \ b) \rightarrow h \ b)$

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 $\text{Ran } g \ h \ a = \text{Ran } (\forall b. (a \rightarrow g \ b) \rightarrow h \ b)$

For example (Danvy & Filinski):

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

The 4-ary type constructor $/\rightarrow/$ 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 $\text{Ran } g \ h \ a = \text{Ran } (\forall b. (a \rightarrow g \ b) \rightarrow h \ b)$

For example (Danvy & Filinski):

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

The 4-ary type constructor $/\rightarrow/$ 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

$\xi_0 k.$ "Ouch!" : $(\text{int} \rightarrow T) \rightarrow \text{string}$

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

$\xi_0 k.$ "Ouch!" : (int $\uparrow T$) \downarrow string

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

Lacuna

$\xi_0 k. \text{"Ouch!"}$:	$(\text{int} \uparrow T) \downarrow \text{string}$
$1 < 2$:	bool
$1 < 2$:	$(\text{bool} \uparrow T) \downarrow T$
$(\xi_0 k. \text{"Ouch!"}) < 2$:	$(\text{bool} \uparrow T) \downarrow \text{string}$
$(\xi_0 k. \text{"Ouch!"}) < (\xi_0 k. \text{'c'})$:	$(\text{bool} \uparrow T) \downarrow \text{string}$

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

Outline

Delimited continuations

Answer types

► **Types as abstract interpretation**

Substructural logic

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

Big step

Operational semantics

$$\frac{\frac{\frac{}{1 \Downarrow 1} \quad \frac{}{2 \Downarrow 2}}{1 + 2 \Downarrow 3} \quad \frac{\frac{}{3 \Downarrow 3} \quad \frac{}{4 \Downarrow 4}}{3 + 4 \Downarrow 7}}{1 + 2 < 3 + 4 \Downarrow \text{true}}}$$

Abstract interpretation

$$\frac{\frac{\frac{}{1 : \text{int}} \quad \frac{}{2 : \text{int}}}{1 + 2 : \text{int}} \quad \frac{\frac{}{3 : \text{int}} \quad \frac{}{4 : \text{int}}}{3 + 4 : \text{int}}}{1 + 2 < 3 + 4 : \text{bool}}}$$

For control effects, need small steps.

Small step

Operational semantics

$$\frac{\frac{\frac{\overline{\text{true}} \Downarrow \text{true}}{3 < 7 \Downarrow \text{true}}}{3 < 3 + 4 \Downarrow \text{true}}}{1 + 2 < 3 + 4 \Downarrow \text{true}} \quad \begin{array}{c} \uparrow \\ \text{time} \end{array}$$

Abstract interpretation

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

$$\frac{\frac{\overline{\text{bool}} : \text{bool}}{\text{int} < \text{int} : \text{bool}}}{\text{int} < 3 + 4 : \text{bool}}}{1 + 2 < 3 + 4 : \text{bool}}$$

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

Abstract expressions and abstract evaluation contexts

Plugging works in both directions.

$$(U \uparrow T) \$ U \quad : \quad T$$

$$S \$ (S \downarrow T) \quad : \quad T$$

Also need hypothetical reasoning.

Abstract expressions and abstract evaluation contexts

$1 < 2$: bool

Abstract expressions and abstract evaluation contexts

$$\frac{[k : \text{bool} \uparrow T]^1 \quad [z : \text{bool}]^2}{\frac{k \$ z : T}{k \$ (1 < 2) : T}^2}^1$$

$$\frac{}{1 < 2 : (\text{bool} \uparrow T) \downarrow T}^1$$

$$\frac{(\text{bool} \uparrow T) \$ \text{bool} : T}{(\text{bool} \uparrow T) \$ (1 < 2) : T}$$

$$\frac{}{1 < 2 : (\text{bool} \uparrow T) \downarrow T}$$

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

Abstract expressions and abstract evaluation contexts

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

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

Abstract expressions and abstract evaluation contexts

$$\begin{array}{c}
 \frac{}{\text{"Ouch!" : string}} \quad \frac{}{\xi_0 k. \text{"Ouch!"}} \quad \frac{}{\xi_0 k. \text{"Ouch!"} : (int \uparrow char) \downarrow string} \quad \frac{}{\xi_0 k. 'c'} \quad \frac{}{\xi_0 k. 'c' : (int \uparrow T) \downarrow char} \quad \frac{}{\xi_0 k. 'c' : (int \uparrow T) \downarrow char} \quad \frac{}{(bool \uparrow T) \$ bool : T} \quad \frac{}{(bool \uparrow T) \$ (int < int) : T} \\
 \frac{}{(bool \uparrow T) \$ ((\xi_0 k. \text{"Ouch!"}) < (\xi_0 k. 'c')) : string} \quad \frac{}{(\xi_0 k. \text{"Ouch!"}) < (\xi_0 k. 'c') : (bool \uparrow T) \downarrow string} \\
 \frac{}{(\xi_0 k. \text{"Ouch!"}) < (\xi_0 k. 'c') : (bool \uparrow T) \downarrow string}
 \end{array}$$

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.

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

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

$T \rightarrow T'$ is T' / T .

Modes construct syntax

The type of “woman” cannot be $\text{person} \setminus \text{bool}$ or $\text{bool} / \text{person}$.

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

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

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

Modes interact through stipulated *structural rules*.

Modes construct syntax

A binary mode without products

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

A unary mode with products

$$\frac{\langle E \rangle : T}{E : \square \downarrow T} \square \downarrow I \quad \frac{E : \square \downarrow T}{\langle E \rangle : T} \square \downarrow E \quad \frac{E : T}{\langle E \rangle : \diamond T} \diamond I \quad \frac{E : \diamond T \quad \Gamma[\langle T \rangle] : T'}{\Gamma[E] : T'} \diamond E$$

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

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

A unary mode with products

$$\frac{\langle E \rangle : T}{E : \square \downarrow T} \square \downarrow I \quad \frac{E : \square \downarrow T}{\langle E \rangle : T} \square \downarrow E \quad \frac{E : T}{\langle E \rangle : \diamond T} \diamond I \quad \frac{E : \diamond T \quad \Gamma[\langle T \rangle] : T'}{\Gamma[E] : T'} \diamond E$$

For delimited continuations, reify evaluation contexts as coterms.

We add two binary modes for coterms:	E, C	$C[[\]E]$
	$C; V$	$C[V[\]]$
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

$$C \$ FE = E, C \$ F$$

$$C \$ VE = C; V \$ E$$

$$V = # \$ V$$

$$\frac{\Gamma[C \$ FE] : T}{\Gamma[E, C \$ F] : T}$$

$$\frac{\Gamma[C \$ VE] : T}{\Gamma[C; V \$ E] : T}$$

$$\frac{\Gamma[V] : T}{\Gamma[# \$ V] : T}$$

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

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 λ_{ξ_0} -calculus with small-step typing**

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	$U ::= U \rightarrow T \mid \text{string} \mid \text{int} \mid \dots$
Cotypes	$S ::= U \uparrow T$
Transitions	

$$C_1 \$ \dots \$ C_n \$ (\lambda x.E)V \quad \rightsquigarrow C_1 \$ \dots \$ C_n \$ E\{x \mapsto V\}$$

$$C_1 \$ \dots \$ C_n \$ C \$ (\xi_0 k.E) \rightsquigarrow C_1 \$ \dots \$ C_n \$ E\{k \mapsto C\}$$

Reset: dynamic semantics

Alternate between refocusing and reducing.

\$ "Goldilocks said: " ^
 (# \$ "This porridge is " ^ "too hot" ^ ". ")
= #; ("Goldilocks said: " ^) \$
 (#; ("This porridge is " ^) \$ "too hot" ^ ". ")
~> #; ("Goldilocks said: " ^) \$
 (#; ("This porridge is " ^) \$ "too hot. ")
= #; ("Goldilocks said: " ^) \$ (# \$ "This porridge is " ^ "too hot. ")
~> #; ("Goldilocks said: " ^) \$ (# \$ "This porridge is too hot. ")
= # \$ "Goldilocks said: " ^ "This porridge is too hot. "
~> # \$ "Goldilocks said: This porridge is too hot. "
= "Goldilocks said: This porridge is too hot. "

Shift₀: dynamic semantics

\$ "Goldilocks said: " \wedge
(# \$ "This porridge is " \wedge
($\xi_0 k.(k $ "too hot") \wedge (k $ "too cold") \wedge (k $ "just right"))$
 \wedge ". ")

= #; ("Goldilocks said: " \wedge) \$
((" . ", (#; ("This porridge is " \wedge))); \wedge) \$
($\xi_0 k.(k $ "too hot") \wedge (k $ "too cold") \wedge (k $ "just right"))$)

\rightsquigarrow #; ("Goldilocks said: " \wedge) \$
(((((" . ", (#; ("This porridge is " \wedge))); \wedge) \$ "too hot") \wedge
((((" . ", (#; ("This porridge is " \wedge))); \wedge) \$ "too cold") \wedge
((((" . ", (#; ("This porridge is " \wedge))); \wedge) \$ "just right"))

= #; ("Goldilocks said: " \wedge) \$
((# \$ "This porridge is " \wedge "too hot" \wedge ". ") \wedge
(# \$ "This porridge is " \wedge "too cold" \wedge ". ") \wedge
(# \$ "This porridge is " \wedge "just right" \wedge ". ")

$\rightsquigarrow \dots$

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

$U ::= U \rightarrow T \mid \text{string} \mid \text{int} \mid \dots$

Cotypes

$S ::= U \uparrow T$

$$\frac{[x : U] \vdots E : T}{\lambda x. E : U \rightarrow T} \lambda$$

$$\frac{[k : S] \vdots E : T}{\xi_0 k. E : S \downarrow T} \xi_0$$

$$\frac{[x : U] \vdots F : U \quad E[x] : T}{E[F] : T} E[U]$$

$$\frac{[x : U] \vdots Vx : T}{V : U \rightarrow T} \rightarrow I$$

$$\frac{[k : S] \vdots k \$ E : T}{E : S \downarrow T} \downarrow I$$

$$\frac{[x : U] \vdots C \$ x : T}{C : U \uparrow T} \uparrow I$$

$$\frac{F : U \rightarrow T \quad E : U}{FE : T} \rightarrow E$$

$$\frac{C : S \quad E : S \downarrow T}{C \$ E : T} \downarrow E$$

$$\frac{C : U \uparrow T \quad E : U}{C \$ E : T} \uparrow E$$

Reset: static semantics

Compute cotype by hypothetical reasoning.

$$\frac{\begin{array}{c} [x : \text{string}] \\ \vdots \\ \# \$ (\text{"Tpi"} \frown x \frown \text{"."}) : \text{string} \end{array}}{\begin{array}{c} ((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) \$ x : \text{string} \\ \hline ((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) : \text{string} \uparrow \text{string} \end{array}} \uparrow I \frac{\text{"too hot"} : \text{string}}{\begin{array}{c} ((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) \$ \text{"too hot"} : \text{string} \\ \hline \# \$ (\text{"Tpi"} \frown \text{"too hot"} \frown \text{"."}) : \text{string} \\ \vdots \end{array}} = \uparrow E$$

Shift₀: static semantics

Compute cotype by hypothetical reasoning.

$$\begin{array}{c}
 [x : \text{string}] \\
 \vdots \\
 \frac{\# \$ (\text{"Tpi"} \frown x \frown \text{"."}) : \text{string}}{((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) \$ x : \text{string}} = \\
 \frac{((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) : \text{string} \uparrow \text{string}}{((\text{"."}, (\#; (\text{"Tpi"} \frown))) ; \frown) \$ \text{"too hot"} : \text{string}} \uparrow \text{I} \quad \begin{array}{c} \vdots \\ \xi_0 k. (k \$ \text{"too hot"}) \frown \dots \\ : (\text{string} \uparrow \text{string}) \downarrow \text{string} \end{array} \\
 \frac{\# \$ (\text{"Tpi"} \frown \text{"too hot"} \frown \text{"."}) : \text{string}}{\vdots} \downarrow \text{E}
 \end{array}$$

Type checking algorithm

$length(\xi_0 k : \text{string} \uparrow \text{bool. 'x'})$

Say that $length$ is a $\text{string} \rightarrow \text{int}$ function.

Type checking algorithm

Implemented in Twelf.

$$\frac{\frac{\frac{[k : \text{string} \uparrow \text{bool}]^1}{\vdots} \quad 'x' : \text{char}}{(\xi_0 k : \text{string} \uparrow \text{bool}. 'x')} \quad 1}{\Rightarrow (\text{string} \uparrow \text{bool}) \downarrow \text{char}} \quad \frac{\frac{[x : \text{string}]^2}{\text{length } x \Rightarrow \text{int}} \quad \frac{[n : \text{int}]^3}{\text{int} \uparrow \text{bool} \Leftarrow \langle \rangle \dot{\smile} n : \text{bool}}}{\text{int} \uparrow \text{bool} \Leftarrow \langle \rangle \dot{\smile} \text{length } x : \text{bool}} \quad 3}{\text{int} \uparrow \text{bool} \Leftarrow \langle \rangle ; \text{length} \dot{\smile} x : \text{bool}} \quad 2}{\langle \rangle ; \text{length} \dot{\smile} (\xi_0 k : \text{string} \uparrow \text{bool}. 'x') \Rightarrow (\text{int} \uparrow \text{bool}) \downarrow \text{char}} \quad \frac{\langle \rangle \dot{\smile} \text{length}(\xi_0 k : \text{string} \uparrow \text{bool}. 'x') \Rightarrow (\text{int} \uparrow \text{bool}) \downarrow \text{char}}{\quad}$$

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