

# Higher-ranked Exception Types

WORK-IN-PROGRESS

Ruud Koot

Department of Information and Computing Sciences  
Utrecht University

March 26, 2015

# Motivation

- ▶ Types should not lie; we would like to have *checked exceptions* in Haskell:

$$\text{map} :: (\alpha \rightarrow \beta) \rightarrow [\alpha] \rightarrow [\beta] \text{ throws } e$$

- ▶ What should be the correct value of  $e$ ?

# Motivation

Assigning accurate exception types is complicated by:

**Higher-order functions** Exceptions raised by higher-order functions depend on the exceptions raised by functional arguments.

$$\text{map} :: (\alpha \rightarrow \beta \text{ throws } e_1) \rightarrow [\alpha] \rightarrow [\beta] \text{ throws } (e_1 \cup e_2)$$

**Non-strict evaluation** Exceptions are embedded inside values.

$$\begin{aligned} \text{map} :: & (\alpha \text{ throws } e_1 \rightarrow \beta) \text{ throws } e_2 \\ & \rightarrow [\alpha \text{ throws } e_3] \text{ throws } e_4 \rightarrow [\beta \text{ throws } e_5] \text{ throws } e_6 \end{aligned}$$

# Motivation

- ▶ Instead of  $\tau$  **throws**  $e$ , write  $\tau\langle e \rangle$  for the exception type of a term of type  $\tau$  that can evaluate to  $\perp_\chi$  for some exception label  $\chi \in e$ .
- ▶ The fully annotated exception type for *map* would be:

$$\begin{aligned} \text{map} :: (\alpha\langle e_1 \rangle \rightarrow \beta\langle e_1 \cup e_2 \rangle)\langle e_3 \rangle &\rightarrow [\alpha\langle e_1 \rangle]\langle e_4 \rangle \\ &\rightarrow [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle \end{aligned}$$

$$\begin{aligned} \text{map} &= \lambda f. \lambda xs. \text{case } xs \text{ of} \\ &\quad [] \quad \quad \mapsto [] \\ &\quad (y : ys) \mapsto f \ y : \text{map } f \ ys \end{aligned}$$

# Motivation

- ▶ Instead of  $\tau$  **throws**  $e$ , write  $\tau\langle e \rangle$  for the exception type of a term of type  $\tau$  that can evaluate to  $\perp_\chi$  for some exception label  $\chi \in e$ .
- ▶ The fully annotated exception type for *map* would be:

$$\begin{aligned} \text{map} :: (\alpha\langle e_1 \rangle \rightarrow \beta\langle e_1 \cup e_2 \rangle)\langle e_3 \rangle &\rightarrow [\alpha\langle e_1 \rangle]\langle e_4 \rangle \\ &\rightarrow [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle \end{aligned}$$

$$\begin{aligned} \text{map} &= \lambda f. \lambda xs. \text{case } xs \text{ of} \\ &\quad [] \quad \mapsto [] \\ &\quad (y : ys) \mapsto f \ y : \text{map } f \ ys \end{aligned}$$

- ▶ If you want to be pedantic:

$$\begin{aligned} \text{map} :: \forall \alpha \ \beta \ e_1 \ e_2 \ e_3 \ e_4. &((\alpha\langle e_1 \rangle \rightarrow \beta\langle e_1 \cup e_2 \rangle)\langle e_3 \rangle \\ &\rightarrow ([\alpha\langle e_1 \rangle]\langle e_4 \rangle \rightarrow [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle)\langle \emptyset \rangle)\langle \emptyset \rangle \end{aligned}$$

# Motivation

- ▶ Instead of  $\tau$  **throws**  $e$ , write  $\tau\langle e \rangle$  for the exception type of a term of type  $\tau$  that can evaluate to  $\perp_\chi$  for some exception label  $\chi \in e$ .
- ▶ The fully annotated exception type for *map* would be:

$$\begin{aligned} \text{map} :: (\alpha\langle e_1 \rangle \rightarrow \beta\langle e_1 \cup e_2 \rangle)\langle e_3 \rangle &\rightarrow [\alpha\langle e_1 \rangle]\langle e_4 \rangle \\ &\rightarrow [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle \end{aligned}$$

$$\begin{aligned} \text{map} &= \lambda f. \lambda xs. \text{case } xs \text{ of} \\ &\quad [] \quad \quad \mapsto [] \\ &\quad (y : ys) \mapsto f \ y : \text{map } f \ ys \end{aligned}$$

- ▶ If you want to be pedantic:

$$\begin{aligned} \text{map} :: \forall \alpha \ \beta \ e_1 \ e_2 \ e_3 \ e_4. \\ (\alpha\langle e_1 \rangle \xrightarrow{e_3} \beta\langle e_1 \cup e_2 \rangle) \xrightarrow{\emptyset} [\alpha\langle e_1 \rangle]\langle e_4 \rangle \xrightarrow{\emptyset} [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle \end{aligned}$$

# Motivation

- ▶ The exception type

$$(\alpha\langle e_1 \rangle \xrightarrow{e_3} \beta\langle e_1 \cup e_2 \rangle) \rightarrow [\alpha\langle e_1 \rangle]\langle e_4 \rangle \rightarrow [\beta\langle e_1 \cup e_2 \cup e_3 \rangle]\langle e_4 \rangle$$

is not as accurate as we would like.

- ▶ Consider the instantiations:

$$\begin{aligned} \text{map id} &:: [\alpha\langle e_1 \rangle]\langle e_4 \rangle \rightarrow [\alpha\langle e_1 \rangle]\langle e_4 \rangle \\ \text{map (const } \perp_{\mathbf{E}}) &:: [\alpha\langle e_1 \rangle]\langle e_4 \rangle \rightarrow [\beta\langle e_1 \cup \{\mathbf{E}\} \rangle]\langle e_4 \rangle \end{aligned}$$

- ▶ A more appropriate type for  $\text{map (const } \perp_{\mathbf{E}})$  would be

$$\text{map (const } \perp_{\mathbf{E}}) :: [\alpha\langle e_1 \rangle]\langle e_4 \rangle \rightarrow [\beta\langle \{\mathbf{E}\} \rangle]\langle e_4 \rangle$$

as it cannot propagate exceptional elements inside the input list to the output list.

# Motivation

- ▶ The problem is that we have already committed the first argument of *map* to be of type

$$\alpha\langle e_1 \rangle \xrightarrow{e_3} \beta\langle e_1 \cup e_2 \rangle,$$

i.e. it propagates exceptional values from the its input to the output while possibly adding additional exceptional values.

- ▶ This is a worst-case scenario: it is sound but inaccurate.



# Motivation

- ▶ The solution is to move from Hindley–Milner to  $F_\omega$ , introducing *higher-ranked types* and *type operators*.
  - ▶ Recall that System  $F_\omega$  replicates the *simply typed  $\lambda$ -calculus* on the type level.
- ▶ This gives us the expressiveness to state the exception type of *map* as:

$$\begin{aligned} & \forall e_2 \, e_3. (\forall e_1. \alpha \langle e_1 \rangle \xrightarrow{e_3} \beta \langle e_2 \, e_1 \rangle) \\ & \rightarrow (\forall e_4 \, e_5. [\alpha \langle e_4 \rangle] \langle e_5 \rangle \rightarrow [\beta \langle e_2 \, e_4 \cup e_3 \rangle] \langle e_5 \rangle) \end{aligned}$$

- ▶ Note that  $e_2$  is an *exception operator* of kind  $\text{EXN} \Rightarrow \text{EXN}$ .

# Motivation

- ▶ Given the following functions:

$$\begin{aligned} \text{map} \quad &:: \forall e_2 e_3. (\forall e_1. \alpha \langle e_1 \rangle \xrightarrow{e_3} \beta \langle e_2 e_1 \rangle) \\ &\rightarrow (\forall e_4 e_5. [\alpha \langle e_4 \rangle] \langle e_5 \rangle \rightarrow [\beta \langle e_2 e_4 \cup e_3 \rangle] \langle e_5 \rangle) \end{aligned}$$

$$\text{id} \quad :: \forall e. \alpha \langle e \rangle \xrightarrow{\emptyset} \alpha \langle e \rangle$$

$$\text{const } \perp_{\mathbf{E}} :: \forall e. \alpha \langle e \rangle \xrightarrow{\emptyset} \beta \langle \{\mathbf{E}\} \rangle$$

- ▶ Applying *id* or *const*  $\perp_{\mathbf{E}}$  to *map* will give rise the the instantiations  $e_2 \mapsto \lambda e. e$ , respectively  $e_2 \mapsto \lambda e. \{\mathbf{E}\}$ .
- ▶ This gives us the exception types:

$$\text{map id} \quad :: \forall e_4 e_5. [\alpha \langle e_4 \rangle] \langle e_5 \rangle \rightarrow [\alpha \langle e_4 \rangle] \langle e_5 \rangle$$

$$\text{map (const } \perp_{\mathbf{E}}) :: \forall e_4 e_5. [\alpha \langle e_4 \rangle] \langle e_5 \rangle \rightarrow [\beta \langle \{\mathbf{E}\} \rangle] \langle e_5 \rangle$$

as desired.

# Intermezzo: Simply-typed $\lambda$ -calculus

## Types

$$\begin{array}{lll} \tau \in \mathbf{Ty} & ::= & B \quad \text{(base type)} \\ & | & \tau_1 \rightarrow \tau_2 \quad \text{(function type)} \end{array}$$

## Terms

$$\begin{array}{lll} t \in \mathbf{Tm} & ::= & x, y, \dots \quad \text{(variable)} \\ & | & \lambda x : \tau. t \quad \text{(abstraction)} \\ & | & t_1 t_2 \quad \text{(application)} \end{array}$$

## Values

$$\begin{array}{lll} v \in \mathbf{Val} & ::= & x, y, \dots \quad \text{(free variable)} \\ & | & \lambda x : \tau. v \quad \text{(abstraction value)} \end{array}$$

# Intermezzo: Simply-typed $\lambda$ -calculus

## Typing

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \text{[T-VAR]} \qquad \frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x : \tau_1. t : \tau_1 \rightarrow \tau_2} \text{[T-ABS]}$$
$$\frac{\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1 t_2 : \tau_2} \text{[T-APP]}$$

# Intermezzo: Simply-typed $\lambda$ -calculus

**Evaluation** We perform *full  $\beta$ -reduction*, i.e. we also evaluate under binders.

$$\frac{t \longrightarrow t'}{\lambda x : \tau. t \longrightarrow \lambda x : \tau. t'} \text{ [E-ABS]}$$

$$\frac{t_1 \longrightarrow t'_1}{t_1 \ t_2 \longrightarrow t'_1 \ t_2} \text{ [E-APP}_1\text{]} \qquad \frac{t_2 \longrightarrow t'_2}{t_1 \ t_2 \longrightarrow t_1 \ t'_2} \text{ [E-APP}_2\text{]}$$

$$\frac{}{(\lambda x : \tau. t_1) \ t_2 \longrightarrow [t_2/x] \ t_1} \text{ [E-BETA]}$$

# Intermezzo: Simply-typed $\lambda$ -calculus

## Theorem (Progress)

*A term  $t$  is either a value  $v$ , or we can reduce  $t \longrightarrow t'$ .*

## Theorem (Preservation)

*If  $\Gamma \vdash t : \tau$  and  $t \longrightarrow t'$ , then  $\Gamma \vdash t' : \tau$ .*

## Theorem (Confluence)

*If  $t \longrightarrow t_1$  and  $t \longrightarrow t_2$ , then exists a term  $t'$  such that  $t_1 \longrightarrow^* t'$  and  $t_2 \longrightarrow^* t'$ .*

## Theorem (Normalization)

*For any term  $t$  we have that  $t \longrightarrow^* v$  (in a finite number of steps).*

## Corollary (Uniqueness of normal forms)

*If  $t \longrightarrow^* v_1$  and  $t \longrightarrow^* v_2$ , then  $v_1 \equiv v_2$ .*

## Intermezzo: The lambda “cube”

- ▶ The simply-typed  $\lambda$ -calculus can be extended with *parametric polymorphism*, or *type operators*, or both.

$$\begin{array}{ccc} F & \longrightarrow & F_\omega \\ \uparrow & & \uparrow \\ \lambda & \longrightarrow & \lambda_\omega \end{array}$$

- |  |  |
|--|--|
| ▶ $id : B \rightarrow B$<br>$id = \lambda x : B. x$  | ▶ $id : \forall \alpha :: *. \alpha \rightarrow \alpha$<br>$id = \Lambda \alpha : *. \lambda x : \alpha. x$  |
| ▶ $Id :: * \Rightarrow *$<br>$Id = \lambda \alpha :: *. \alpha$<br>$id : B \rightarrow Id\ B$<br>$id = \lambda x : B. x$ | ▶ $Id :: * \Rightarrow *$<br>$Id = \lambda \alpha :: *. \alpha$<br>$id : \forall \alpha :: *. \alpha \rightarrow Id\ \alpha$<br>$id = \Lambda \alpha : *. \lambda x : \alpha. x$ |

- ▶ Omitted: the axis for dependent types.

# Intermezzo: System $F_\omega$

## Types

$\tau \in \mathbf{Ty}$	$::=$	$\alpha$	(type variable)
	$ $	$\tau_1 \rightarrow \tau_2$	(function type)
	$ $	$\forall \alpha :: \kappa. \tau$	(universal type)
	$ $	$\lambda \alpha :: \kappa. \tau$	(operator abstraction)
	$ $	$\tau_1 \tau_2$	(operator application)

## Kinds

$\kappa \in \mathbf{Kind}$	$::=$	$*$	(base kind)
	$ $	$\kappa_1 \Rightarrow \kappa_2$	(operator kind)



# Intermezzo: System $F_\omega$

## Terms

$t \in \mathbf{Tm}$	$::=$	$x, y, \dots$	(variable)
		$\lambda x : \tau. t$	(abstraction)
		$t_1 \ t_2$	(application)
		$\Lambda \alpha :: \kappa. t$	(type abstraction)
		$t \ \langle \tau \rangle$	(type application)

## Values

$v \in \mathbf{Val}$	$::=$	$x, y, \dots$	(free variable)
		$\lambda x : \tau. v$	(abstraction value)
		$\Lambda \alpha : \kappa. v$	(type abstraction value)

## Intermezzo: System $F_\omega$

**Kinding** Note the similarity between the *type* system of the simply typed  $\lambda$ -calculus.

$$\frac{}{\Gamma, \alpha :: \kappa \vdash \alpha :: \kappa} [\text{K-VAR}] \qquad \frac{\Gamma, \alpha :: \kappa_1 \vdash \tau_2 :: \kappa_2}{\Gamma \vdash \lambda \alpha :: \kappa_1. \tau_2 :: \kappa_1 \Rightarrow \kappa_2} [\text{K-ABS}]$$

$$\frac{\Gamma \vdash \tau_1 :: \kappa_1 \Rightarrow \kappa_2 \quad \Gamma \vdash \tau_2 :: \kappa_1}{\Gamma \vdash \tau_1 \tau_2 :: \kappa_2} [\text{K-APP}]$$

$$\frac{\Gamma \vdash \tau_1 :: * \quad \Gamma \vdash \tau_2 :: *}{\Gamma \vdash \tau_1 \rightarrow \tau_2 :: *} [\text{K-ARROW}] \qquad \frac{\Gamma, \alpha :: \kappa \vdash \tau :: *}{\Gamma \vdash \forall \alpha :: \kappa. \tau :: *} [\text{K-FORALL}]$$

# Intermezzo: System $F_\omega$

## Type equivalence

$$\frac{}{\tau \equiv \tau} \text{ [Q-REFL]}$$

$$\frac{\tau_1 \equiv \tau_2}{\tau_2 \equiv \tau_1} \text{ [Q-SYMM]}$$

$$\frac{\tau_1 \equiv \tau_2 \quad \tau_2 \equiv \tau_3}{\tau_1 \equiv \tau_3} \text{ [Q-TRANS]}$$

$$\frac{\tau_1 \equiv \tau'_1 \quad \tau_2 \equiv \tau'_2}{\tau_1 \rightarrow \tau_2 \equiv \tau'_1 \rightarrow \tau'_2} \text{ [Q-ARROW]}$$

$$\frac{\tau_1 \equiv \tau_2}{\forall \alpha :: \kappa. \tau_1 \equiv \forall \alpha :: \kappa. \tau_2} \text{ [Q-FORALL]}$$

$$\frac{\tau_1 \equiv \tau_2}{\lambda \alpha :: \kappa. \tau_1 \equiv \lambda \alpha :: \kappa. \tau_2} \text{ [Q-ABS]}$$

$$\frac{\tau_1 \equiv \tau'_1 \quad \tau_2 \equiv \tau'_2}{\tau_1 \tau_2 \equiv \tau'_1 \tau'_2} \text{ [Q-APP]}$$

$$\frac{}{(\lambda \alpha :: \kappa. \tau_1) \tau_2 \equiv [\alpha \mapsto \tau_2] \tau_1} \text{ [Q-BETA]}$$

## Intermezzo: System $F_\omega$

**Typing** In addition to the rules for the simply typed  $\lambda$ -calculus:

$$\frac{\Gamma \vdash \tau_1 :: * \quad \Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x : \tau_1. t : \tau_1 \rightarrow \tau_2} [\text{T-ABS}]$$

$$\frac{\Gamma, \alpha :: \kappa_1 \vdash t_2 : \tau_2}{\Gamma \vdash \Lambda \alpha :: \kappa_1. t_2 : \forall \alpha :: \kappa_1. \tau_2} [\text{T-TYABS}]$$

$$\frac{\Gamma \vdash t_1 : \forall \alpha :: \kappa_1. \tau_1 \quad \Gamma \vdash \tau_2 :: \kappa_1}{\Gamma \vdash t_1 \langle \tau_2 \rangle : [\alpha \mapsto \tau_2] \tau_1} [\text{T-TYAPP}]$$

$$\frac{\Gamma \vdash t : \tau_1 \quad \tau_1 \equiv \tau_2 \quad \Gamma \vdash \tau_2 :: *}{\Gamma \vdash t : \tau_2} [\text{T-EQ}]$$

## Intermezzo: System $F_\omega$

**Evaluation** In addition to the rules for the simply typed  $\lambda$ -calculus:

$$\frac{t \longrightarrow t'}{\Lambda\alpha : \kappa.t \longrightarrow \Lambda\alpha : \kappa.t'} \text{ [E-TYABS]} \qquad \frac{t \longrightarrow t'}{t \langle \tau \rangle \longrightarrow t' \langle \tau \rangle} \text{ [E-TYAPP]}$$

$$\frac{}{(\Lambda\alpha : \kappa.t) \langle \tau \rangle \longrightarrow [\tau/\alpha] t} \text{ [E-TYBETA]}$$

# Intermezzo: System $F_\omega$

## Metatheory

- ▶ We still have *progress*, *preservation* and *decidability* (of type checking).
- ▶ Proofs rely on the structure of the types and type equivalence relation and thus the properties (especially *normalization* and *uniqueness of normal forms*) of the simply typed  $\lambda$ -calculus.

# Technicalities

- ▶ Due to their syntactic weight, higher-ranked exception type only seem useful if they can be inferred automatically.
- ▶ Unlike for HM type inference is undecidable in  $F_\omega$ .
- ▶ However, the exception types are annotations piggybacking on top of an underlying type system.
- ▶ Holdermans and Hage [HH10] showed type inference is decidable for a higher-ranked annotated type system with type operators performing control-flow analysis.

# Technicalities

1. Perform Hindley–Milner type inference to reconstruct the underlying types.
2. Run a second inference pass to reconstruct the exception types.
  - 2.1 In case of a  $\lambda$ -abstraction  $\lambda x : \tau.e$ , we *complete* the type  $\tau$  to an exception type.
  - 2.2 In case of an application we *match* the types of the formal and actual parameter.



## Technicalities: Reconstruction (variables, constants)

$$\mathcal{R} : \mathbf{TyEnv} \times \mathbf{KiEnv} \times \mathbf{Tm} \rightarrow \mathbf{ExnTy} \times \mathbf{Exn}$$

$$\mathcal{R} \Gamma \Delta x = \Gamma_x$$

$$\mathcal{R} \Gamma \Delta c_\tau = \langle \perp_\tau; \emptyset \rangle$$

$$\mathcal{R} \Gamma \Delta \downarrow_\tau^\ell = \langle \perp_\tau; \{\ell\} \rangle$$

# Technicalities: Reconstruction (abstractions)

$\mathcal{R} : \mathbf{TyEnv} \times \mathbf{KiEnv} \times \mathbf{Tm} \rightarrow \mathbf{ExnTy} \times \mathbf{Exn}$

$\mathcal{R} \Gamma \Delta (\lambda x : \tau. t) =$

**let**  $\langle \hat{\tau}_1; e_1; \overline{e_i : \kappa_i} \rangle = \mathcal{C} \ \emptyset \ \tau$

$\langle \hat{\tau}_2; \varphi_2 \rangle = \mathcal{R} (\Gamma, x : \hat{\tau}_1 \ \& \ e_1) (\Delta, \overline{e_i : \kappa_i}) \ t$

**in**  $\langle \forall \overline{e_i : \kappa_i}. \hat{\tau}_1 \langle e_1 \rangle \rightarrow \hat{\tau}_2 \langle \varphi_2 \rangle; \emptyset \rangle$

# Technicalities: Completion

- The completion procedure adds as many quantifiers and type operators as possible to a type.

$$\frac{}{\overline{e_i :: \kappa_i} \vdash \mathbf{bool} : \widehat{\mathbf{bool}} \ \& \ e \ \overline{e_i} \triangleright e :: \kappa_i \Rightarrow_{\text{EXN}}} \text{[C-Bool]}$$

$$\frac{\overline{e_i :: \kappa_i} \vdash \tau : \hat{\tau} \ \& \ \varphi \triangleright \overline{e_j :: \kappa_j}}{\overline{e_i :: \kappa_i} \vdash [\tau] : [\hat{\tau}(\varphi)] \ \& \ e \ \overline{e_i} \triangleright e :: \kappa_i \Rightarrow_{\text{EXN}}, \overline{e_j :: \kappa_j}} \text{[C-List]}$$

$$\frac{\vdash \tau_1 : \hat{\tau}_1 \ \& \ \varphi_1 \triangleright \overline{e_j :: \kappa_j} \quad \overline{e_i :: \kappa_i}, \overline{e_j :: \kappa_j} \vdash \tau_2 : \hat{\tau}_2 \ \& \ \varphi_2 \triangleright \overline{e_j :: \kappa_j}}{\overline{e_i :: \kappa_i} \vdash \tau_1 \rightarrow \tau_2 : \forall \overline{e_j :: \kappa_j}. (\hat{\tau}_1(\varphi_1) \rightarrow \hat{\tau}_2(\varphi_2)) \ \& \ e \ \overline{e_i} \triangleright e :: \kappa_i \Rightarrow_{\text{EXN}}, \overline{e_k :: \kappa_k}} \text{[C-Arr]}$$

Figure: Type completion ( $\Gamma \vdash \tau : \hat{\tau} \ \& \ \varphi \triangleright \Gamma'$ )

# Technicalities: Completion

- ▶  $\cdot \vdash \mathbf{bool} : \widehat{\mathbf{bool}} \ \& \ e_1 \triangleright e_1 :: \mathbf{EXN}$

# Technicalities: Completion

- ▶  $\cdot \vdash \mathbf{bool} : \widehat{\mathbf{bool}} \ \& \ e_1 \triangleright e_1 :: \mathbf{EXN}$
- ▶  $e_1 :: \mathbf{EXN} \vdash \mathbf{bool} : \widehat{\mathbf{bool}} \ \& \ e_2 \ e_1 \triangleright e_2 :: \mathbf{EXN} \Rightarrow \mathbf{EXN}$

# Technicalities: Completion

- ▶ **bool**  $\rightarrow$  **bool**
- ▶  $\forall e_1 :: \text{EXN}. \widehat{\text{bool}}\langle e_1 \rangle \rightarrow \widehat{\text{bool}}\langle e_2 \ e_1 \rangle \ \& \ e_3$
- ▶  $e_2 :: \text{EXN} \Rightarrow \text{EXN}, e_3 :: \text{EXN}$

# Technicalities: Completion

► **bool** → **bool** → **bool**

►

$$\forall e_1 :: \text{EXN}. \widehat{\text{bool}}\langle e_1 \rangle \rightarrow$$
$$(\forall e_4 :: \text{EXN}. \widehat{\text{bool}}\langle e_4 \rangle \xrightarrow{e_2 \ e_1} \widehat{\text{bool}}\langle e_5 \ e_1 \ e_4 \rangle) \ \& \ e_3$$

►  $e_2 :: \text{EXN} \Rightarrow \text{EXN}, e_3 :: \text{EXN}, e_5 :: \text{EXN} \Rightarrow \text{EXN} \Rightarrow \text{EXN}$

# Technicalities: Completion

► **(bool  $\rightarrow$  bool)  $\rightarrow$  bool**

►

$$\forall e_2 :: \text{EXN} \Rightarrow \text{EXN}. \forall e_3 :: \text{EXN}.$$

$$\left( \forall e_1 :: \text{EXN}. \widehat{\text{bool}}\langle e_1 \rangle \xrightarrow{e_3} \widehat{\text{bool}}\langle e_2 \ e_1 \rangle \right) \rightarrow \widehat{\text{bool}}\langle e_4 \ e_2 \ e_3 \rangle \ \& \ e_5$$

►  $e_4 :: (\text{EXN} \Rightarrow \text{EXN}) \Rightarrow \text{EXN} \Rightarrow \text{EXN}, e_5 :: \text{EXN}$



# Technicalities: Reconstruction (applications)

$\mathcal{R} : \mathbf{TyEnv} \times \mathbf{KiEnv} \times \mathbf{Tm} \rightarrow \mathbf{ExnTy} \times \mathbf{Exn}$

$\mathcal{R} \Gamma \Delta (t_1 \ t_2) =$

**let**  $\langle \hat{\tau}_1; \varphi_1 \rangle \quad \quad \quad = \mathcal{R} \Gamma \Delta t_1$

$\langle \hat{\tau}_2; \varphi_2 \rangle \quad \quad \quad = \mathcal{R} \Gamma \Delta t_2$

$\langle \hat{\tau}'_2 \langle e'_2 \rangle \rightarrow \hat{\tau}' \langle \varphi' \rangle; \overline{e_i : \kappa_i} \rangle = \mathcal{I} \hat{\tau}_1$

$\theta \quad \quad \quad = [e'_2 \mapsto \varphi_2] \circ \mathcal{M} \oslash \hat{\tau}_2 \ \hat{\tau}'_2$

**in**  $\langle \llbracket \theta \hat{\tau}' \rrbracket_{\Delta}; \llbracket \theta \varphi' \cup \varphi_1 \rrbracket_{\Delta} \rangle$

# Technicalities: Matching

$\mathcal{M} :: \mathbf{Env} \rightarrow \mathbf{T}_y \rightarrow \mathbf{T}_y \rightarrow \mathbf{Subst}$

$\mathcal{M} \Sigma \widehat{\mathbf{bool}} \quad \widehat{\mathbf{bool}} \quad = Id$

$\mathcal{M} \Sigma (\forall e :: \kappa. \widehat{\tau}_1) \quad (\forall e :: \kappa. \widehat{\tau}'_1) \quad = \mathcal{M} (\Sigma, e \mapsto \kappa) \widehat{\tau}_1 \widehat{\tau}'_1$

$\mathcal{M} \Sigma (\widehat{\tau}_1^{e_1} \rightarrow \widehat{\tau}_2^{\chi_2}) (\widehat{\tau}_1^{e_1} \rightarrow \widehat{\tau}_2'^{(e_0 \bar{e}_j)}) =$

$\left[ e_0 \mapsto (\overline{\lambda e_j :: \Sigma(e_j). \chi_2}) \right] \circ \mathcal{M} \Sigma \widehat{\tau}_2 \widehat{\tau}'_2$

$\mathcal{M} \Sigma \_ \quad \_ \quad = \mathbf{fail}$

# Technicalities: Matching — Example

- ▶  $\mathcal{M} [e_1 :: \text{EXN}, e_2 :: \text{EXN} \Rightarrow \text{EXN}, e_3 :: \text{EXN}]$   
     $(\widehat{\text{bool}}\langle e_1 \rangle \rightarrow \widehat{\text{bool}}\langle e_2 \ e_1 \ \cup \ e_3 \rangle)$   
     $(\widehat{\text{bool}}\langle e_1 \rangle \rightarrow \widehat{\text{bool}}\langle e_0 \ e_1 \ e_2 \ e_3 \rangle)$

# Technicalities: Matching — Example

- ▶  $\mathcal{M} [e_1 :: \text{EXN}, e_2 :: \text{EXN} \Rightarrow \text{EXN}, e_3 :: \text{EXN}]$   
     $(\widehat{\text{bool}}\langle e_1 \rangle \rightarrow \widehat{\text{bool}}\langle e_2 \ e_1 \ \cup \ e_3 \rangle)$   
     $(\widehat{\text{bool}}\langle e_1 \rangle \rightarrow \widehat{\text{bool}}\langle e_0 \ e_1 \ e_2 \ e_3 \rangle)$
- ▶  $[e_0 \mapsto \lambda e_1 :: \text{EXN}.\lambda e_2 :: \text{EXN} \Rightarrow \text{EXN}.\lambda e_3 :: \text{EXN}.e_2 \ e_1 \ \cup \ e_3])$

# Technicalities: Reconstruction (fixpoints)

$\mathcal{R} : \mathbf{TyEnv} \times \mathbf{KiEnv} \times \mathbf{Tm} \rightarrow \mathbf{ExnTy} \times \mathbf{Exn}$

$\mathcal{R} \Gamma \Delta (\mathbf{fix} \ t) =$

**let**  $\langle \hat{\tau}; \varphi \rangle \quad \quad \quad = \mathcal{R} \Gamma \Delta \ t$

$\langle \hat{\tau}' \langle e' \rangle \rightarrow \hat{\tau}'' \langle \varphi'' \rangle; \overline{e_i : \kappa_i} \rangle = \mathcal{I} \ \hat{\tau}$

**in**  $\langle \hat{\tau}_0; \varphi_0; i \rangle \leftarrow \langle \perp_{[\hat{\tau}']} ; \emptyset; 0 \rangle$

**do**  $\theta \quad \quad \quad \leftarrow [e' \mapsto \varphi_i] \circ \mathcal{M} \ \emptyset \ \hat{\tau}_i \ \hat{\tau}'$

$\langle \hat{\tau}_{i+1}; \varphi_{i+1}; i \rangle \leftarrow \langle \llbracket \theta \hat{\tau}'' \rrbracket_{\Delta}; \llbracket \theta \varphi'' \rrbracket_{\Delta}; i+1 \rangle$

**until**  $\langle \hat{\tau}_i; \varphi_i \rangle \equiv \langle \hat{\tau}_{i-1}; \varphi_{i-1} \rangle$

**return**  $\langle \hat{\tau}_i; \llbracket \varphi \cup \varphi_i \rrbracket_{\Delta} \rangle$

# Technicalities: $\lambda^U$

## Types

$\tau \in \mathbf{Ty}$	$::= \mathcal{P}$	(base type)
	$  \tau_1 \rightarrow \tau_2$	(function type)

## Terms

$t \in \mathbf{Tm}$	$::= x, y, \dots$	(variable)
	$  \lambda x : \tau. t$	(abstraction)
	$  t_1 t_2$	(application)
	$  \emptyset$	(empty)
	$  \{c\}$	(singleton)
	$  t_1 \cup t_2$	(union)

**Values** Values  $v$  are terms of the form

$$\lambda x_1 : \tau_1. \dots \lambda x_i : \tau_i. \{c_1\} \cup (\dots \cup (\{c_j\} \cup (x_1 v_{11} \dots v_{1m} \cup (\dots \cup x_k v_{k1} \dots v_{kn}))))))$$

# Technicalities: $\lambda^\cup$

$$(\lambda x : \tau. t_1) \ t_2 \longrightarrow [t_2/x] \ t_1 \quad (\beta\text{-reduction})$$

$$(t_1 \cup t_2) \ t_3 \longrightarrow t_1 \ t_3 \cup t_2 \ t_3$$

$$(\lambda x : \tau. t_1) \cup (\lambda x : \tau. t_2) \longrightarrow \lambda x : \tau. (t_1 \cup t_2) \quad (\text{congruences})$$

$$x \ t_1 \cdots t_n \cup x' \ t'_1 \cdots t'_n \longrightarrow x \ (t_1 \cup t'_1) \cdots (t_n \cup t'_n)$$

$$(t_1 \cup t_2) \cup t_3 \longrightarrow t_1 \cup (t_2 \cup t_3) \quad (\text{associativity})$$

$$\emptyset \cup t \longrightarrow t$$

$$t \cup \emptyset \longrightarrow t$$

(unit)

$$x \cup x \longrightarrow x$$

$$x \cup (x \cup t) \longrightarrow x \cup t$$

(idempotence)

$$\{c\} \cup \{c\} \longrightarrow \{c\}$$

$$\{c\} \cup (\{c\} \cup t) \longrightarrow \{c\} \cup t$$

$$x \ t_1 \cdots t_n \cup \{c\} \longrightarrow \{c\} \cup x \ t_1 \cdots t_n \quad (1)$$

$$x \ t_1 \cdots t_n \cup (\{c\} \cup t) \longrightarrow \{c\} \cup (x \ t_1 \cdots t_n \cup t) \quad (2)$$

$$x \ t_1 \cdots t_n \cup x' \ t'_1 \cdots t'_n \longrightarrow x' \ t'_1 \cdots t'_n \cup x \ t_1 \cdots t_n \quad \text{if } x' \prec x \quad (3)$$

$$x \ t_1 \cdots t_n \cup (x' \ t'_1 \cdots t'_n \cup t) \longrightarrow x' \ t'_1 \cdots t'_n \cup (x \ t_1 \cdots t_n \cup t) \quad \text{if } x' \prec x \quad (4)$$

$$\{c\} \cup \{c'\} \longrightarrow \{c'\} \cup \{c\} \quad \text{if } c' \prec c \quad (5)$$

$$\{c\} \cup (\{c'\} \cup t) \longrightarrow \{c'\} \cup (\{c\} \cup t) \quad \text{if } c' \prec c \quad (6)$$

# Technicalities: $\lambda^U$

## Conjecture

*The reduction relation  $\longrightarrow$  preserves meaning.*

## Conjecture

*The reduction relation  $\longrightarrow$  is strongly normalizing.*

## Conjecture

*The reduction relation  $\longrightarrow$  is locally confluent.*

## Corollary

*The reduction relation  $\longrightarrow$  is confluent.*

## Corollary

*The  $\lambda^U$ -calculus has unique normal forms.*

## Corollary

*Equality of  $\lambda^U$ -terms can be decided by normalization.*



# Problems

- ▶ Not sound w.r.t. *imprecise exception semantics*.
- ▶ Making it sound negates the precision gained by higher-ranked types.
- ▶ Need to move to a more powerful language for effects.
  - ▶ In previous work we used conditional constraints/implications and a somewhat ad hoc non-emptiness guard.
  - ▶ Now I want to look at *Boolean rings*, which look more well-behaved.

## Problems: Imprecise exception semantics

- ▶ In an optimizing compiler we want the following equality, called the *case-switching transformation*, to hold:

$$\begin{aligned} \forall e_i. & \text{if } e_1 \text{ then} \\ & \quad \text{if } e_2 \text{ then } e_3 \text{ else } e_4 \\ & \text{else} \\ & \quad \text{if } e_2 \text{ then } e_5 \text{ else } e_6 \equiv \text{if } e_2 \text{ then} \\ & \quad \quad \text{if } e_1 \text{ then } e_3 \text{ else } e_5 \\ & \quad \text{else} \\ & \quad \quad \text{if } e_1 \text{ then } e_4 \text{ else } e_6 \end{aligned}$$

- ▶ This does not hold if we have observable exceptions and track them precisely.
  - ▶ Counterexample: Take  $e_1 = \perp_{E_1}$  and  $e_2 = \perp_{E_2}$ .
- ▶ Introduce some “imprecision”: If the guard can reduce to an exceptional value, then pretend both branches get executed.

## Problems: Imprecise exception semantics

- ▶ In an optimizing compiler we want the following equality, called the *case-switching transformation*, to hold:

$$\begin{aligned} \forall e_i. & \text{if } e_1 \text{ then} \\ & \quad \text{if } e_2 \text{ then } e_3 \text{ else } e_4 \\ & \text{else} \\ & \quad \text{if } e_2 \text{ then } e_5 \text{ else } e_6 \equiv \text{if } e_2 \text{ then} \\ & \quad \quad \text{if } e_1 \text{ then } e_3 \text{ else } e_5 \\ & \quad \text{else} \\ & \quad \quad \text{if } e_1 \text{ then } e_4 \text{ else } e_6 \end{aligned}$$

- ▶ This does not hold if we have observable exceptions and track them precisely.
  - ▶ Counterexample: Take  $e_1 = \perp_{E_1}$  and  $e_2 = \perp_{E_2}$ .
- ▶ Introduce some “imprecision”: **If** the guard can reduce to an exceptional value, **then** pretend both branches get executed.



# References

## Paper and prototype

A prototype implementation of the type inference algorithm and an early draft of the paper are available from:

- <https://github.com/ruudkoot/phd>

## Related work

-  Stefan Holdermans and Jurriaan Hage, *Polyvariant flow analysis with higher-ranked polymorphic types and higher-order effect operators*, Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming, ICFP '10, 2010, pp. 63–74.
-  Andrew J. Kennedy, *Type inference and equational theories*, Tech. Report LIX-RR-96-09, Laboratoire D'Informatique, École Polytechnique, 1996.