Principles of Program Analysis:

Type and Effect Systems

Transparencies based on Chapter 5 of the book: Flemming Nielson, Hanne Riis Nielson and Chris Hankin: Principles of Program Analysis. Springer Verlag 2005.

Basic idea: effect systems

If an expression e maps entities of type τ_1 to entities of type τ_2

$$e : \tau_1 \to \tau_2$$

then we can annotate the arrow with properties of the program

$$e : \tau_1 \xrightarrow{\varphi} \tau_2$$

Example analysis	Choice of the property φ of a function call
Control Flow	which function abstractions might arise
Side Effect	which side effects might be observed
Exception	which exceptions might be raised
Region	which regions of data might be effected
Communication	which temporal behaviour might be observed

The plan

- a typed functional language
- with a traditional underlying type system
- several extensions to effect systems:

Analysis	characteristica	properties
Control Flow	subeffecting	sets
Side Effect	subtyping	sets
Exception	polymorphism	sets
Region	polymorphic recursion	sets
Communication	polymorphism	temporal

Syntax of the Fun language

$$e \;::=\; c \mid x \mid \texttt{fn}_{\pi}\; x \;\texttt{=>}\; e_0 \mid \texttt{fun}_{\pi}\; f\; x \;\texttt{=>}\; e_0 \mid e_1\; e_2$$

$\uparrow$ $\uparrow$

program points

$$\mid \quad \texttt{if}\; e_0\; \texttt{then}\; e_1\; \texttt{else}\; e_2 \mid \underbrace{\texttt{let}\; x\; \texttt{=}\; e_1\; \texttt{in}\; e_2}_{\text{not polymorphic}} \mid e_1\; op\; e_2$$

Examples:

- `(fn`$_X$ `x => x) (fn`$_Y$ `y => y)`
- `let g = (fun`$_F$ `f x => f (fn`$_Y$ `y => y))`
  `in g (fn`$_Z$ `z => z)`

Underlying type system: typing judgements

$$\Gamma \vdash_{\mathsf{UL}} e : \tau$$

$$\tau ::= \mathtt{int} \mid \mathtt{bool} \mid \tau_1 \to \tau_2$$

$$\Gamma ::= [\ ] \mid \Gamma[x \mapsto \tau]$$

Assumptions:

- each constant c has a type τ_c

  $\mathtt{true}$ has type $\tau_{\mathtt{true}} = \mathtt{bool}$; 7 has type $\tau_7 = \mathtt{int}$

- each operator *op* expects two arguments of type τ_{op}^1 and τ_{op}^2 and gives a result of type τ_{op}

  $>$ expects two arguments of type $\mathtt{int}$ and gives a result of type $\mathtt{bool}$

Underlying type system: axioms and rules (1)

$$\Gamma \vdash_{\mathsf{UL}} c : \tau_c$$

$$\Gamma \vdash_{\mathsf{UL}} x : \tau \qquad \text{if } \Gamma(x) = \tau$$

$$\frac{\Gamma[x \mapsto \tau_x] \vdash_{\mathsf{UL}} e_0 : \tau_0}{\Gamma \vdash_{\mathsf{UL}} \mathbf{fn}_\pi\ x\ \texttt{=>}\ e_0 : \tau_x \to \tau_0}$$

$$\frac{\Gamma[f \mapsto \tau_x \to \tau_0][x \mapsto \tau_x] \vdash_{\mathsf{UL}} e_0 : \tau_0}{\Gamma \vdash_{\mathsf{UL}} \mathbf{fun}_\pi\ f\ x\ \texttt{=>}\ e_0 : \tau_x \to \tau_0}$$

$$\frac{\Gamma \vdash_{\mathsf{UL}} e_1 : \tau_2 \to \tau_0 \qquad \Gamma \vdash_{\mathsf{UL}} e_2 : \tau_2}{\Gamma \vdash_{\mathsf{UL}} e_1\ e_2 : \tau_0}$$

Underlying type system: axioms and rules (2)

$$\frac{\Gamma \vdash_{\mathsf{UL}} e_0 : \mathtt{bool} \quad \Gamma \vdash_{\mathsf{UL}} e_1 : \tau \quad \Gamma \vdash_{\mathsf{UL}} e_2 : \tau}{\Gamma \vdash_{\mathsf{UL}} \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau}$$

$$\frac{\Gamma \vdash_{\mathsf{UL}} e_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash_{\mathsf{UL}} e_2 : \tau_2}{\Gamma \vdash_{\mathsf{UL}} \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 : \tau_2}$$

$$\frac{\Gamma \vdash_{\mathsf{UL}} e_1 : \tau_{op}^1 \quad \Gamma \vdash_{\mathsf{UL}} e_2 : \tau_{op}^2}{\Gamma \vdash_{\mathsf{UL}} e_1\ \mathsf{op}\ e_2 : \tau_{op}}$$

Example:

```
let g = (fun_F f x => f (fn_Y y => y))
in g (fn_Z z => z)
```

Abbreviation: $\Gamma_{\mathsf{fx}} = [\mathtt{f} \mapsto (\tau \to \tau) \to (\tau \to \tau)][\mathtt{x} \mapsto \tau \to \tau]$

Inference tree:

$$\dfrac{\Gamma_{\mathsf{fx}} \vdash_{\mathsf{UL}} \mathtt{f} : (\tau \to \tau) \to (\tau \to \tau) \qquad \dfrac{\Gamma_{\mathsf{fx}}[\mathtt{y} \mapsto \tau] \vdash_{\mathsf{UL}} \mathtt{y} : \tau}{\Gamma_{\mathsf{fx}} \vdash_{\mathsf{UL}} \mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y} : \tau \to \tau}}{\dfrac{\Gamma_{\mathsf{fx}} \vdash_{\mathsf{UL}} \mathtt{f}\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}) : \tau \to \tau}{[\,] \vdash_{\mathsf{UL}} \mathtt{fun}_{\mathsf{F}}\ \mathtt{f}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{f}\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}) : (\tau \to \tau) \to (\tau \to \tau)}}$$

Control Flow Analysis

The aim of the analysis:

> For each subexpression, which function abstractions might it evaluate to?

Values of type `int` and `bool` can only evaluate to integers and booleans

Values of type $\tau_1 \to \tau_2$ can only evaluate to function abstractions

- annotate the arrow with the program points for these abstractions

Example: $\texttt{fn}_{\mathsf{X}}\ \texttt{x => x} : \texttt{int} \xrightarrow{\{\mathsf{X}\}} \texttt{int}$

$\texttt{fn}_{\mathsf{X}}\ \texttt{x => x} : \texttt{int} \xrightarrow{\{\mathsf{X},\mathsf{Y}\}} \texttt{int}$ subeffecting

Control Flow Analysis: typing judgements

$$\widehat{\Gamma} \vdash_{\mathsf{CFA}} e : \widehat{\tau}$$

$$\widehat{\tau} ::= \texttt{int} \mid \texttt{bool} \mid \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2$$

$$\widehat{\Gamma} ::= [\ ] \mid \widehat{\Gamma}[x \mapsto \widehat{\tau}] \qquad \varphi ::= \{\pi\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset$$

Back to the underlying type system: remove the annotations

$$\lfloor \texttt{int} \rfloor = \texttt{int} \qquad \lfloor \texttt{bool} \rfloor = \texttt{bool}$$

$$\lfloor \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \rfloor = \lfloor \widehat{\tau}_1 \rfloor \to \lfloor \widehat{\tau}_2 \rfloor$$

For type environments: $\lfloor \widehat{\Gamma} \rfloor (x) = \lfloor \widehat{\Gamma}(x) \rfloor$ for all x

Control Flow Analysis: axioms and rules (1)

$$\widehat{\Gamma} \;\vdash_{\text{CFA}}\; c : \tau_c$$

$$\widehat{\Gamma} \;\vdash_{\text{CFA}}\; x : \widehat{\tau} \qquad\qquad \text{if } \widehat{\Gamma}(x) = \widehat{\tau}$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \;\vdash_{\text{CFA}}\; e_0 : \widehat{\tau}_0}{\widehat{\Gamma} \;\vdash_{\text{CFA}}\; \mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\{\pi\}\cup\boxed{\varphi}} \tau_0}$$

subeffecting

$$\frac{\widehat{\Gamma}[f \mapsto \widehat{\tau}_x \xrightarrow{\{\pi\}\cup\boxed{\varphi}} \widehat{\tau}_0][x \mapsto \widehat{\tau}_x] \;\vdash_{\text{CFA}}\; e_0 : \widehat{\tau}_0}{\widehat{\Gamma} \;\vdash_{\text{CFA}}\; \mathtt{fun}_\pi\ f\ x\ \mathtt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\{\pi\}\cup\boxed{\varphi}} \widehat{\tau}_0}$$

$$\frac{\widehat{\Gamma} \;\vdash_{\text{CFA}}\; e_1 : \widehat{\tau}_2 \xrightarrow{\boxed{\varphi}} \widehat{\tau}_0 \qquad \widehat{\Gamma} \;\vdash_{\text{CFA}}\; e_2 : \widehat{\tau}_2}{\widehat{\Gamma} \;\vdash_{\text{CFA}}\; e_1\ e_2 : \widehat{\tau}_0}$$

Control Flow Analysis: axioms and rules (2)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CFA}} e_0 : \mathtt{bool} \quad \widehat{\Gamma} \vdash_{\mathsf{CFA}} e_1 : \widehat{\tau} \quad \widehat{\Gamma} \vdash_{\mathsf{CFA}} e_2 : \widehat{\tau}}{\widehat{\Gamma} \vdash_{\mathsf{CFA}} \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \widehat{\tau}}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CFA}} e_1 : \widehat{\tau}_1 \quad \widehat{\Gamma}[x \mapsto \widehat{\tau}_1] \vdash_{\mathsf{CFA}} e_2 : \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\mathsf{CFA}} \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 : \widehat{\tau}_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CFA}} e_1 : \tau_{op}^1 \quad \widehat{\Gamma} \vdash_{\mathsf{CFA}} e_2 : \tau_{op}^2}{\widehat{\Gamma} \vdash_{\mathsf{CFA}} e_1\ \mathit{op}\ e_2 : \tau_{op}}$$

Example (1)

```
let g = (fun_F f x => f (fn_Y y => y))
in g (fn_Z z => z)
```

Abbreviation: $\widehat{\Gamma}_{\mathsf{fx}} = [\mathtt{f} \mapsto (\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}) \xrightarrow{\{\mathsf{F}\}} (\widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau})][\mathtt{x} \mapsto \widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}]$

Inference tree:

$$\dfrac{\widehat{\Gamma}_{\mathsf{fx}} \vdash_{\mathsf{CFA}} \mathtt{f} : (\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}) \xrightarrow{\{\mathsf{F}\}} (\widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau}) \qquad \dfrac{\widehat{\Gamma}_{\mathsf{fx}}[\mathtt{y} \mapsto \widehat{\tau}] \vdash_{\mathsf{CFA}} \mathtt{y} : \widehat{\tau}}{\widehat{\Gamma}_{\mathsf{fx}} \vdash_{\mathsf{CFA}} \mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y} : \widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}}}{\dfrac{\widehat{\Gamma}_{\mathsf{fx}} \vdash_{\mathsf{CFA}} \mathtt{f}\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}) : \widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau}}{[\,] \vdash_{\mathsf{CFA}} \mathtt{fun}_{\mathsf{F}}\ \mathtt{f}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{f}\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}) : (\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}) \xrightarrow{\{\mathsf{F}\}} (\widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau})}}$$

Example (2)

```
let g = (fun_F f x => f (fn_Y y => y))
in g (fn_Z z => z)
```

Abbreviation: $\widehat{\Gamma}_{\mathrm{g}} = [\mathrm{g} \mapsto (\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}) \xrightarrow{\{\mathsf{F}\}} (\widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau})]$

Inference tree:

$$\dfrac{\widehat{\Gamma}_{\mathrm{g}} \vdash_{\mathsf{CFA}} \mathtt{g} : (\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}) \xrightarrow{\{\mathsf{F}\}} (\widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau}) \qquad \dfrac{\widehat{\Gamma}_{\mathrm{g}}[\mathtt{z} \mapsto \widehat{\tau}] \vdash_{\mathsf{CFA}} \mathtt{z} : \widehat{\tau}}{\widehat{\Gamma}_{\mathrm{g}} \vdash_{\mathsf{CFA}} \mathtt{fn}_{\mathsf{Z}}\ \mathtt{z} \Rightarrow \mathtt{z} : \widehat{\tau} \xrightarrow{\{\mathsf{Z},\mathsf{Y}\}} \widehat{\tau}}}{\widehat{\Gamma}_{\mathrm{g}} \vdash_{\mathsf{CFA}} \mathtt{g}\ (\mathtt{fn}_{\mathsf{Z}}\ \mathtt{z} \Rightarrow \mathtt{z}) : \widehat{\tau} \xrightarrow{\emptyset} \widehat{\tau}}$$

the program never terminates

$\boxed{\text{assuming } \{\mathsf{Y},\mathsf{Z}\} = \{\mathsf{Z},\mathsf{Y}\}}$

Example:

Abbreviation: $\widehat{\tau}_{\mathsf{Y}} = \mathtt{int} \xrightarrow{\{\mathsf{Y}\}} \mathtt{int}$

Inference tree:

$$\dfrac{\dfrac{[\mathtt{x} \mapsto \widehat{\tau}_{\mathsf{Y}}] \vdash_{\mathsf{CFA}} \mathtt{x} : \widehat{\tau}_{\mathsf{Y}}}{[\,] \vdash_{\mathsf{CFA}} \mathtt{fn}_{\mathsf{X}}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{x} : \widehat{\tau}_{\mathsf{Y}} \xrightarrow{\{\mathsf{X}\}} \widehat{\tau}_{\mathsf{Y}}} \qquad \dfrac{[\mathtt{y} \mapsto \mathtt{int}] \vdash_{\mathsf{CFA}} \mathtt{y} : \mathtt{int}}{[\,] \vdash_{\mathsf{CFA}} \mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y} : \widehat{\tau}_{\mathsf{Y}}}}{[\,] \vdash_{\mathsf{CFA}} (\mathtt{fn}_{\mathsf{X}}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{x})\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}) : \widehat{\tau}_{\mathsf{Y}}}$$

Note: the whole inference tree is needed to get full information about the control flow properties.

Some subtleties

- formally we should write $\{\pi_1\} \cup \cdots \cup \{\pi_n\}$ but we write $\{\pi_1, \cdots, \pi_n\}$

- we can replace $\tau_1 \xrightarrow{\varphi_1} \tau_2$ by $\tau_1 \xrightarrow{\varphi_2} \tau_2$ whenever φ_1 and φ_2 are "equal as sets"

$$\varphi = \varphi \cup \emptyset \qquad \varphi = \varphi \cup \varphi$$

$$\varphi_1 \cup \varphi_2 = \varphi_2 \cup \varphi_1 \qquad \varphi_1 \cup (\varphi_2 \cup \varphi_3) = (\varphi_1 \cup \varphi_2) \cup \varphi_3$$

$$\varphi = \varphi \qquad \frac{\varphi_1 = \varphi_2 \quad \varphi_2 = \varphi_3}{\varphi_1 = \varphi_3} \qquad \frac{\varphi_1 = \varphi_1' \quad \varphi_2 = \varphi_2'}{\varphi_1 \cup \varphi_2 = \varphi_1' \cup \varphi_2'}$$

- we can replace $\widehat{\tau}_1$ by $\widehat{\tau}_2$ if they have the same underlying types and all annotations on corresponding function arrows are "equal as sets"

$$\widehat{\tau} = \widehat{\tau} \qquad \frac{\widehat{\tau}_1 = \widehat{\tau}_1' \qquad \widehat{\tau}_2 = \widehat{\tau}_2' \qquad \varphi = \varphi'}{(\widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2) = (\widehat{\tau}_1' \xrightarrow{\varphi'} \widehat{\tau}_2')}$$

One more subtlety

The function `fn`$_{\mathsf{Y}}$ `y => y` has type $\boxed{\widehat{\tau} \xrightarrow{\{\mathsf{Y},\mathsf{Z}\}} \widehat{\tau}}$ as well as $\boxed{\widehat{\tau} \xrightarrow{\{\mathsf{Y}\}} \widehat{\tau}}$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \ \vdash_{\mathsf{CFA}} \ e_0 : \widehat{\tau}_0}{\widehat{\Gamma} \ \vdash_{\mathsf{CFA}} \ \texttt{fn}_\pi \ x \ \texttt{=>} \ e_0 : \widehat{\tau}_x \xrightarrow{\{\pi\} \cup \boxed{\varphi}} \tau_0}$$

Conservative extension lemma

(i) If $\widehat{\Gamma} \ \vdash_{\mathsf{CFA}} \ e : \widehat{\tau}$ then $\lfloor\widehat{\Gamma}\rfloor \vdash_{\mathsf{UL}} e : \lfloor\widehat{\tau}\rfloor$.

(ii) If $\Gamma \vdash_{\mathsf{UL}} e : \tau$ then there exists $\widehat{\Gamma}$ and $\widehat{\tau}$ such that $\widehat{\Gamma} \ \vdash_{\mathsf{CFA}} \ e : \widehat{\tau}$, $\lfloor\widehat{\Gamma}\rfloor = \Gamma$ and $\lfloor\widehat{\tau}\rfloor = \tau$.

If we replaced the above rule by

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \vdash'_{\mathsf{CFA}} e_0 : \widehat{\tau}_0}{\widehat{\Gamma} \vdash'_{\mathsf{CFA}} \texttt{fn}_\pi \ x \ \texttt{=>} \ e_0 : \widehat{\tau}_x \xrightarrow{\{\pi\}} \tau_0}$$

then some programs would have no type in the Control Flow Analysis!

Operational Semantics

Different choices:

- Structural Operational Semantics
- Natural Semantics
  - with environments
  - with substitutions

Assumption: e is a *closed* expression; it evaluates to a value v

$$v ::= c \mid \mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0 \qquad\qquad \text{(closed expressions only)}$$

written $\boxed{\vdash e \longrightarrow v}$

Natural Semantics for Fun (1)

$$\vdash c \longrightarrow c$$

$$\vdash (\mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0) \longrightarrow (\mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0)$$

$$\vdash (\mathtt{fun}_\pi\ f\ x\ \mathtt{=>}\ e_0) \longrightarrow (\mathtt{fn}_\pi\ x\ \mathtt{=>}\ (e_0[f \mapsto \mathtt{fun}_\pi\ f\ x\ \mathtt{=>}\ e_0]))$$

$$\frac{\vdash e_1 \longrightarrow (\mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0) \quad \vdash e_2 \longrightarrow v_2 \quad \vdash e_0[x \mapsto v_2] \longrightarrow v_0}{\vdash e_1\ e_2 \longrightarrow v_0}$$

Natural Semantics for Fun (2)

$$\frac{\vdash e_0 \longrightarrow \texttt{true} \quad \vdash e_1 \longrightarrow v_1}{\vdash \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 \longrightarrow v_1}$$

$$\frac{\vdash e_0 \longrightarrow \texttt{false} \quad \vdash e_2 \longrightarrow v_2}{\vdash \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 \longrightarrow v_2}$$

$$\frac{\vdash e_1 \longrightarrow v_1 \quad \vdash e_2[x \mapsto v_1] \longrightarrow v_2}{\vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \longrightarrow v_2}$$

$$\frac{\vdash e_1 \longrightarrow v_1 \quad \vdash e_2 \longrightarrow v_2}{\vdash e_1 \; op \; e_2 \longrightarrow v} \quad \text{if } v_1 \; \mathbf{op} \; v_2 = v$$

Example:

Expression: $(\texttt{fn}_X\ \texttt{x => x})\ (\texttt{fn}_Y\ \texttt{y => y})$

We have

$$\vdash \texttt{fn}_X\ \texttt{x => x} \longrightarrow \texttt{fn}_X\ \texttt{x => x}$$

$$\vdash \texttt{fn}_Y\ \texttt{y => y} \longrightarrow \texttt{fn}_Y\ \texttt{y => y}$$

$$\vdash \underbrace{\texttt{x}[\texttt{x} \mapsto \texttt{fn}_Y\ \texttt{y => y}]}_{\texttt{fn}_Y\ \texttt{y => y}} \longrightarrow \texttt{fn}_Y\ \texttt{y => y}$$

The application rule gives

$$\vdash (\texttt{fn}_X\ \texttt{x => x})\ (\texttt{fn}_Y\ \texttt{y => y}) \longrightarrow \texttt{fn}_Y\ \texttt{y => y}$$

Example:

Expression: `let g = (fun`$_F$` f x => f (fn`$_Y$` y => y))`
`in g (fn`$_Z$` z => z)`

We have

$$\vdash \texttt{fun}_\texttt{F}\ \texttt{f x => f (fn}_\texttt{Y}\ \texttt{y => y)} \longrightarrow$$
$$\texttt{fn}_\texttt{F}\ \texttt{x => ((fun}_\texttt{F}\ \texttt{f x => f(fn}_\texttt{Y}\ \texttt{y => y)) (fn}_\texttt{Y}\ \texttt{y => y))}$$

For the body of the `let`-construct we replace the occurrence of `g` with

$$\texttt{fn}_\texttt{F}\ \texttt{x => ((fun}_\texttt{F}\ \texttt{f x => f (fn}_\texttt{Y}\ \texttt{y => y)) (fn}_\texttt{Y}\ \texttt{y => y))}$$

The operator evaluates to this value and the operand `fn`$_Z$` z => z` evaluates to itself.

The next step is to determine a value v such that

$$\vdash \texttt{(fun}_\texttt{F}\ \texttt{f x => f (fn}_\texttt{Y}\ \texttt{y => y)) (fn}_\texttt{Y}\ \texttt{y => y)} \longrightarrow v$$

and we enter a circularity!

Semantic Correctness

Assumption: If $[\,]\ \vdash_{\mathsf{CFA}}\ v_1 : \tau^1_{op}$ and $[\,]\ \vdash_{\mathsf{CFA}}\ v_2 : \tau^2_{op}$ and $v = v_1\ \mathbf{op}\ v_2$ then $[\,]\ \vdash_{\mathsf{CFA}}\ v : \tau_{op}$.

Theorem: If $[\,]\ \vdash_{\mathsf{CFA}}\ e : \widehat{\tau}$, and $\vdash e \longrightarrow v$ then $[\,]\ \vdash_{\mathsf{CFA}}\ v : \widehat{\tau}$.

Consequences:

- if $[\,] \vdash e : \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2$ and $\vdash e \longrightarrow \texttt{fn}_\pi\ x\ \texttt{=>}\ e_0$ then $\pi \in \varphi$
- if $[\,] \vdash e : \widehat{\tau}_1 \xrightarrow{\emptyset} \widehat{\tau}_2$ then e cannot terminate!

Auxiliary results needed for correctness proof

- If $\widehat{\Gamma}_1 \vdash_{\mathsf{CFA}} e : \widehat{\tau}$ and $\forall x \in \mathit{FV}(e) : \widehat{\Gamma}_1(x) = \widehat{\Gamma}_2(x)$

  then $\widehat{\Gamma}_2 \vdash_{\mathsf{CFA}} e : \widehat{\tau}$.

- If $[\,] \vdash_{\mathsf{CFA}} e_0 : \widehat{\tau}_0$ and $\widehat{\Gamma}[x \mapsto \widehat{\tau}_0] \vdash_{\mathsf{CFA}} e : \widehat{\tau}$

  then $\widehat{\Gamma} \vdash_{\mathsf{CFA}} e[x \mapsto e_0] : \widehat{\tau}$.

Important questions

- can all programs be analysed?

- does there always exist a best analysis result?

Can we establish a Moore family result?

Complete lattice of annotations

$(\mathbf{Ann}, \sqsubseteq)$ is a complete lattice isomorphic to $(\mathcal{P}(\mathbf{Pnt}), \subseteq)$

Complete lattice of annotated types

$(\widehat{\mathbf{Type}}[\tau], \sqsubseteq)$ is the complete lattice with

- elements: annotated types $\widehat{\tau}$ with underlying type τ (i.e. $\lfloor\widehat{\tau}\rfloor = \tau$)

- ordering defined by

$$\widehat{\tau} \sqsubseteq \widehat{\tau} \qquad \frac{\widehat{\tau}_1 \sqsubseteq \widehat{\tau}_1' \quad \varphi \subseteq \varphi' \quad \widehat{\tau}_2 \sqsubseteq \widehat{\tau}_2'}{\widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \sqsubseteq \widehat{\tau}_1' \xrightarrow{\varphi'} \widehat{\tau}_2'}$$

Example: $(\texttt{int} \xrightarrow{\varphi_1} \texttt{int}) \xrightarrow{\varphi_2} \texttt{int} \ \sqsubseteq \ (\texttt{int} \xrightarrow{\varphi_3} \texttt{int}) \xrightarrow{\varphi_4} \texttt{int}$ will be the case if and only if $\varphi_1 \subseteq \varphi_3$ and $\varphi_2 \subseteq \varphi_4$. (Note the covariance.)

Moore family result

Define

$$\mathsf{JUDG}_{\mathsf{CFA}}[\Gamma \vdash_{\mathsf{UL}} e : \tau]$$

to be the set of typings $\widehat{\Gamma} \vdash_{\mathsf{CFA}} e : \widehat{\tau}$ such that $\lfloor \widehat{\Gamma} \vdash_{\mathsf{CFA}} e : \widehat{\tau} \rfloor = \Gamma \vdash_{\mathsf{UL}} e : \tau$

Then $\mathsf{JUDG}_{\mathsf{CFA}}[\Gamma \vdash_{\mathsf{UL}} e : \tau]$ is a Moore family whenever $\Gamma \vdash_{\mathsf{UL}} e : \tau$.

Implementation

- type reconstruction algorithm for the underlying type system;

  unification procedure for underlying types

- type reconstruction algorithm for Control Flow Analysis;

  unification procedure for annotated types

- syntactic soundness: whatever the algorithm determines is correct with respect to the specification

- syntactic completeness: if some analysis result is allowed by the specification, then the algorithm will produce it (or something better)

Underlying type system

$$\mathcal{W}_{\mathsf{UL}}(\Gamma, e) = (\tau, \theta)$$

substitution: the modifications needed for Γ

$\theta : \mathbf{TypVar} \rightarrow_{\mathsf{fin}} \mathbf{Type}$

the type of e: $\tau \in \mathbf{Type}$ $\quad \tau ::= \mathtt{int} \mid \mathtt{bool} \mid \tau_1 \rightarrow \tau_2 \mid \alpha$

$\alpha \in \mathbf{TypVar}$ $\quad \alpha ::= {}'\mathtt{a} \mid {}'\mathtt{b} \mid {}'\mathtt{c} \mid \cdots$

the expression to be analysed

the current type environment: $\Gamma ::= [\,] \mid \Gamma[x \mapsto \tau]$

Idea: if $\mathcal{W}_{\mathsf{UL}}(\Gamma, e) = (\tau, \theta)$ then $\theta_G(\theta\ \Gamma) \vdash_{\mathsf{UL}} e : \theta_G\, \tau$
for all *ground* substitutions θ_G

Type reconstruction algorithm (1)

$$\mathcal{W}_{\mathsf{UL}}(\Gamma, c) = (\tau_c, \quad id)$$

$$\mathcal{W}_{\mathsf{UL}}(\Gamma, x) = (\Gamma(x), \quad id)$$

$$\begin{array}{lll}\mathcal{W}_{\mathsf{UL}}(\Gamma, \mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0) = & \text{let} & \alpha_x \text{ be fresh} \\ & & (\tau_0, \theta_0) = \mathcal{W}_{\mathsf{UL}}(\Gamma[x \mapsto \alpha_x], e_0) \\ & \text{in} & ((\theta_0\ \alpha_x) \to \tau_0, \quad \theta_0)\end{array}$$

$$\begin{array}{lll}\mathcal{W}_{\mathsf{UL}}(\Gamma, \mathtt{fun}_\pi\ f\ x\ \mathtt{=>}\ e_0) = & \text{let} & \alpha_x, \alpha_0 \text{ be fresh} \\ & & (\tau_0, \theta_0) = \mathcal{W}_{\mathsf{UL}}(\Gamma[f \mapsto \alpha_x \to \alpha_0][x \mapsto \alpha_x], e_0) \\ & & \theta_1 = \mathcal{U}_{\mathsf{UL}}(\tau_0, \theta_0\ \alpha_0) \\ & \text{in} & (\theta_1(\theta_0\ \alpha_x) \to \theta_1\ \tau_0, \quad \theta_1 \circ \theta_0)\end{array}$$

$$\begin{array}{lll}\mathcal{W}_{\mathsf{UL}}(\Gamma, e_1\ e_2) = & \text{let} & (\tau_1, \theta_1) = \mathcal{W}_{\mathsf{UL}}(\Gamma, e_1) \\ & & (\tau_2, \theta_2) = \mathcal{W}_{\mathsf{UL}}(\theta_1\ \Gamma, e_2) \\ & & \alpha \text{ be fresh} \\ & & \theta_3 = \mathcal{U}_{\mathsf{UL}}(\theta_2\ \tau_1, \tau_2 \to \alpha) \\ & \text{in} & (\theta_3\ \alpha, \quad \theta_3 \circ \theta_2 \circ \theta_1)\end{array}$$

Type reconstruction algorithm (2)

$$\begin{array}{rll}
\mathcal{W}_{\mathsf{UL}}(\Gamma, \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2) = & \text{let} & (\tau_0, \theta_0) = \mathcal{W}_{\mathsf{UL}}(\Gamma, e_0) \\
& & (\tau_1, \theta_1) = \mathcal{W}_{\mathsf{UL}}(\theta_0\ \Gamma, e_1) \\
& & (\tau_2, \theta_2) = \mathcal{W}_{\mathsf{UL}}(\theta_1(\theta_0\ \Gamma), e_2) \\
& & \theta_3 = \mathcal{U}_{\mathsf{UL}}(\theta_2(\theta_1\ \tau_0), \texttt{bool}) \\
& & \theta_4 = \mathcal{U}_{\mathsf{UL}}(\theta_3\ \tau_2, \theta_3(\theta_2\ \tau_1)) \\
& \text{in} & (\theta_4(\theta_3\ \tau_2),\quad \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1)
\end{array}$$

$$\begin{array}{rll}
\mathcal{W}_{\mathsf{UL}}(\Gamma, \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2) = & \text{let} & (\tau_1, \theta_1) = \mathcal{W}_{\mathsf{UL}}(\Gamma, e_1) \\
& & (\tau_2, \theta_2) = \mathcal{W}_{\mathsf{UL}}((\theta_1 \Gamma)[x \mapsto \tau_1], e_2) \\
& \text{in} & (\tau_2,\quad \theta_2 \circ \theta_1)
\end{array}$$

$$\begin{array}{rll}
\mathcal{W}_{\mathsf{UL}}(\Gamma, e_1\ \mathit{op}\ e_2) = & \text{let} & (\tau_1, \theta_1) = \mathcal{W}_{\mathsf{UL}}(\Gamma, e_1) \\
& & (\tau_2, \theta_2) = \mathcal{W}_{\mathsf{UL}}(\theta_1 \Gamma, e_2) \\
& & \theta_3 = \mathcal{U}_{\mathsf{UL}}(\theta_2\ \tau_1, \tau_{op}^1) \\
& & \theta_4 = \mathcal{U}_{\mathsf{UL}}(\theta_3\ \tau_2, \tau_{op}^2) \\
& \text{in} & (\tau_{op},\quad \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1)
\end{array}$$

Example:

$\mathcal{W}_{\mathsf{UL}}([\,], (\mathtt{fn}_X\ \mathtt{x\ =>\ x})\ (\mathtt{fn}_Y\ \mathtt{y\ =>\ y}))$

call $\mathcal{W}_{\mathsf{UL}}([\,], \mathtt{fn}_X\ \mathtt{x\ =>\ x})$

create the fresh type variable $'\mathrm{a}$ and return $('\mathtt{a} \to {'\mathtt{a}}, \mathit{id})$

call $\mathcal{W}_{\mathsf{UL}}([\,], \mathtt{fn}_Y\ \mathtt{y\ =>\ y})$

create the fresh type variable $'\mathrm{b}$ and return $('\mathtt{b} \to {'\mathtt{b}}, \mathit{id})$

create the fresh type variable $'\mathrm{c}$

call $\mathcal{U}_{\mathsf{UL}}('\mathtt{a} \to {'\mathtt{a}},\ ('\mathtt{b} \to {'\mathtt{b}}) \to {'\mathtt{c}})$ and return $['\mathtt{a} \mapsto {'\mathtt{b}} \to {'\mathtt{b}}]['\mathtt{c} \mapsto {'\mathtt{b}} \to {'\mathtt{b}}]$

return $('\mathtt{b} \to {'\mathtt{b}}, ['\mathtt{a} \mapsto {'\mathtt{b}} \to {'\mathtt{b}}]['\mathtt{c} \mapsto {'\mathtt{b}} \to {'\mathtt{b}}])$

Unification

$$
\begin{array}{rcl}
\mathcal{U}_{\mathsf{UL}}(\mathtt{int}, \mathtt{int}) & = & \mathit{id} \\[2ex]
\mathcal{U}_{\mathsf{UL}}(\mathtt{bool}, \mathtt{bool}) & = & \mathit{id} \\[2ex]
\mathcal{U}_{\mathsf{UL}}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') & = & \begin{array}{ll} \mathsf{let} & \theta_1 = \mathcal{U}_{\mathsf{UL}}(\tau_1, \tau_1') \\ & \theta_2 = \mathcal{U}_{\mathsf{UL}}(\theta_1\ \tau_2, \theta_1\ \tau_2') \\ \mathsf{in} & \theta_2 \circ \theta_1 \end{array} \\[2ex]
\mathcal{U}_{\mathsf{UL}}(\tau, \alpha) & = & \left\{ \begin{array}{ll} [\alpha \mapsto \tau] & \text{if } \alpha \text{ does not occur in } \tau \\ & \text{or if } \alpha \text{ equals } \tau \\ \mathsf{fail} & \text{otherwise} \end{array} \right. \\[2ex]
\mathcal{U}_{\mathsf{UL}}(\alpha, \tau) & = & \left\{ \begin{array}{ll} [\alpha \mapsto \tau] & \text{if } \alpha \text{ does not occur in } \tau \\ & \text{or if } \alpha \text{ equals } \tau \\ \mathsf{fail} & \text{otherwise} \end{array} \right. \\[2ex]
\mathcal{U}_{\mathsf{UL}}(\tau_1, \tau_2) & = & \mathsf{fail} \qquad \text{in all other cases}
\end{array}
$$

Example:

$\mathcal{U}_{\text{UL}}('\mathtt{a} \rightarrow '\mathtt{a},\ ('\mathtt{b} \rightarrow '\mathtt{b}) \rightarrow '\mathtt{c})$

call $\mathcal{U}_{\text{UL}}('\mathtt{a},\ '\mathtt{b} \rightarrow '\mathtt{b})$

return $['\mathtt{a} \mapsto '\mathtt{b} \rightarrow '\mathtt{b}]$

call $\mathcal{U}_{\text{UL}}('\mathtt{b} \rightarrow '\mathtt{b},\ '\mathtt{c})$

return $['\mathtt{c} \mapsto '\mathtt{b} \rightarrow '\mathtt{b}]$

return $['\mathtt{a} \mapsto '\mathtt{b} \rightarrow '\mathtt{b}]['\mathtt{c} \mapsto '\mathtt{b} \rightarrow '\mathtt{b}]$

Towards an algorithm for Control Flow Analysis

Problem: two annotated types *may* be equal even when their syntactic representations are different ($\texttt{int} \xrightarrow{\{X,Y\}} \texttt{int}$ equals $\texttt{int} \xrightarrow{\{Y,X\}} \texttt{int}$)

– the annotated types constitute a *non-free algebra*

– the underlying types constitute a *free algebra*

Idea:

– restrict the form of annotated types to be "simple":
  only annotation variables are allowed on function arrows

– introduce constraints on the values of the annotation variables

We can adapt the unification procedure to work for Control Flow Analysis.

Control Flow Analysis

$$\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e) = (\widehat{\tau}, \theta, C)$$

set of constraints: $\beta \supseteq \varphi$

$\varphi \in \mathbf{Ann}$ $\qquad \varphi ::= \{\pi\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset \mid \beta$

$\beta \in \mathbf{AnnVar}$ $\qquad \beta ::= {}'1 \mid {}'2 \mid {}'3 \mid \cdots$

substitution: the modifications needed for $\widehat{\Gamma}$

$\theta : (\mathbf{TypVar} \cup \mathbf{AnnVar}) \rightarrow_{\mathsf{fin}} (\mathbf{Type} \cup \mathbf{Ann})$

the type of e: $\widehat{\tau} \in \mathbf{Type}$ $\qquad \widehat{\tau} ::= \mathtt{int} \mid \mathtt{bool} \mid \widehat{\tau}_1 \xrightarrow{\beta} \widehat{\tau}_2 \mid \alpha$

$\alpha \in \mathbf{TypVar}$ $\qquad \alpha ::= {}'\mathtt{a} \mid {}'\mathtt{b} \mid {}'\mathtt{c} \mid \cdots$

the expression to be analysed

the current type environment: $\widehat{\Gamma} ::= [\,] \mid \widehat{\Gamma}[x \mapsto \widehat{\tau}]$

Unification of "simple" types

$$\mathcal{U}_{\mathsf{CFA}}(\mathtt{int},\mathtt{int}) = id$$

$$\mathcal{U}_{\mathsf{CFA}}(\mathtt{bool},\mathtt{bool}) = id$$

$$\mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}_1 \xrightarrow{\beta} \widehat{\tau}_2, \widehat{\tau}_1' \xrightarrow{\beta'} \widehat{\tau}_2') = \begin{array}{ll} \text{let} & \boxed{\theta_0 = [\beta' \mapsto \beta]} \\ & \theta_1 = \mathcal{U}_{\mathsf{CFA}}(\theta_0\ \widehat{\tau}_1, \theta_0\ \widehat{\tau}_1') \\ & \theta_2 = \mathcal{U}_{\mathsf{CFA}}(\theta_1\ (\theta_0\ \widehat{\tau}_2), \theta_1\ (\theta_0\ \widehat{\tau}_2')) \\ \text{in} & \theta_2 \circ \theta_1 \circ \theta_0 \end{array}$$

$$\mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}, \alpha) = \begin{cases} [\alpha \mapsto \widehat{\tau}] & \text{if } \alpha \text{ does not occur in } \widehat{\tau} \\ & \text{or if } \alpha \text{ equals } \widehat{\tau} \\ \text{fail} & \text{otherwise} \end{cases}$$

$$\mathcal{U}_{\mathsf{CFA}}(\alpha, \widehat{\tau}) = \begin{cases} [\alpha \mapsto \widehat{\tau}] & \text{if } \alpha \text{ does not occur in } \widehat{\tau} \\ & \text{or if } \alpha \text{ equals } \widehat{\tau} \\ \text{fail} & \text{otherwise} \end{cases}$$

$$\mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}_1, \widehat{\tau}_2) = \text{fail} \quad \text{in all other cases}$$

Example:

$\mathcal{U}_{\mathsf{CFA}}('\mathtt{a} \xrightarrow{'1} '\mathtt{a},\ ('\mathtt{b} \xrightarrow{'2} '\mathtt{b}) \xrightarrow{'3} '\mathtt{c})$

construct $['3 \mapsto '1]$

call $\mathcal{U}_{\mathsf{CFA}}('\mathtt{a},\ '\mathtt{b} \xrightarrow{'2} '\mathtt{b})$

return $['\mathtt{a} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]$

call $\mathcal{U}_{\mathsf{CFA}}('\mathtt{b} \xrightarrow{'2} '\mathtt{b},\ '\mathtt{c})$

return $['\mathtt{c} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]$

return $['3 \mapsto '1]['\mathtt{a} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]['\mathtt{c} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]$

Theoretical properties

The unification algorithm is *syntactically sound*: if it succeeds then it produces a unifying substitution.

The unification algorithm is *syntactically complete*: if there is some way of unifying the two simple types then the algorithm will succeed.

Formally: Let $\widehat{\tau}_1$ and $\widehat{\tau}_2$ be two "simple" types.

- If $\mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}_1, \widehat{\tau}_2) = \theta$ then θ is a "simple" substitution such that $\theta\,\widehat{\tau}_1 = \theta\,\widehat{\tau}_2$.

- If there exists a substitution θ'' such that $\theta''\widehat{\tau}_1 = \theta''\widehat{\tau}_2$ then there exists substitutions θ and θ' such that $\mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}_1, \widehat{\tau}_2) = \theta$ and $\theta'' = \theta' \circ \theta$.

Type reconstruction for Control Flow Analysis (1)

$$\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, c) = (\tau_c,\ \mathit{id},\ \emptyset)$$

$$\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, x) = (\widehat{\Gamma}(x),\ \mathit{id},\ \emptyset)$$

$$\begin{array}{rll}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, \mathtt{fn}_\pi\ x \ \mathtt{=>}\ e_0) = & \text{let} & \alpha_x \text{ be fresh} \\
& & (\widehat{\tau}_0, \theta_0, C_0) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}[x \mapsto \alpha_x], e_0) \\
& & \beta_0 \text{ be fresh} \\
& \text{in} & ((\theta_0\ \alpha_x) \xrightarrow{\beta_0} \widehat{\tau}_0,\quad \theta_0,\quad C_0 \cup \{\beta_0 \supseteq \{\pi\}\})
\end{array}$$

$$\begin{array}{rll}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_1\ e_2) = & \text{let} & (\widehat{\tau}_1, \theta_1, C_1) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_1) \\
& & (\widehat{\tau}_2, \theta_2, C_2) = \mathcal{W}_{\mathsf{CFA}}(\theta_1\ \widehat{\Gamma}, e_2) \\
& & \alpha, \beta \text{ be fresh} \\
& & \theta_3 = \boxed{\mathcal{U}_{\mathsf{CFA}}(\theta_2\ \widehat{\tau}_1, \widehat{\tau}_2 \xrightarrow{\beta} \alpha)} \\
& \text{in} & (\theta_3\ \alpha,\quad \theta_3 \circ \theta_2 \circ \theta_1,\quad \theta_3\ (\theta_2\ C_1) \cup \theta_3\ C_2)
\end{array}$$

Type reconstruction for Control Flow Analysis (2)

$$
\begin{array}{l}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, \mathtt{fun}_\pi\ f\ x\ \mathtt{=>}\ e_0) = \\
\quad \text{let} \quad \alpha_x, \alpha_0, \beta_0 \text{ be fresh} \\
\quad \phantom{\text{let}} \quad (\widehat{\tau}_0, \theta_0, C_0) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}[f \mapsto \alpha_x \xrightarrow{\beta_0} \alpha_0][x \mapsto \alpha_x], e_0) \\
\quad \phantom{\text{let}} \quad \theta_1 = \mathcal{U}_{\mathsf{CFA}}(\widehat{\tau}_0, \theta_0\ \alpha_0) \\
\quad \text{in} \quad (\theta_1(\theta_0\ \alpha_x) \xrightarrow{\theta_1(\theta_0\ \beta_0)} \theta_1\ \widehat{\tau}_0, \quad \theta_1 \circ \theta_0, \\
\quad \phantom{\text{in}} \quad\quad (\theta_1\ C_0) \cup \{\theta_1(\theta_0\ \beta_0) \supseteq \{\pi\}\})
\end{array}
$$

$$
\begin{array}{l}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2) = \\
\quad \text{let} \quad (\widehat{\tau}_0, \theta_0, C_0) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_0) \\
\quad \phantom{\text{let}} \quad (\widehat{\tau}_1, \theta_1, C_1) = \mathcal{W}_{\mathsf{CFA}}(\theta_0\ \widehat{\Gamma}, e_1) \\
\quad \phantom{\text{let}} \quad (\widehat{\tau}_2, \theta_2, C_2) = \mathcal{W}_{\mathsf{CFA}}(\theta_1\ (\theta_0\ \widehat{\Gamma}), e_2) \\
\quad \phantom{\text{let}} \quad \theta_3 = \mathcal{U}_{\mathsf{CFA}}(\theta_2\ (\theta_1\ \widehat{\tau}_0), \mathtt{bool}) \\
\quad \phantom{\text{let}} \quad \theta_4 = \mathcal{U}_{\mathsf{CFA}}(\theta_3\ \widehat{\tau}_2, \theta_3\ (\theta_2\ \tau_1)) \\
\quad \text{in} \quad (\theta_4\ (\theta_3\ \widehat{\tau}_2), \quad \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1 \circ \theta_0, \\
\quad \phantom{\text{in}} \quad\quad \theta_4\ (\theta_3\ (\theta_2\ (\theta_1\ C_0))) \cup \theta_4\ (\theta_3\ (\theta_2\ C_1)) \cup \theta_4\ (\theta_3\ C_2))
\end{array}
$$

Type reconstruction for Control Flow Analysis (3)

$$
\begin{array}{l}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2) = \\
\qquad \begin{array}{ll}
\text{let} & (\widehat{\tau}_1, \theta_1, C_1) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_1) \\
 & (\widehat{\tau}_2, \theta_2, C_2) = \mathcal{W}_{\mathsf{CFA}}((\theta_1\ \widehat{\Gamma})[x \mapsto \widehat{\tau}_1], e_2) \\
\text{in} & (\widehat{\tau}_2, \quad \theta_2 \circ \theta_1, \quad (\theta_2\ C_1) \cup C_2)
\end{array}
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_1\ \mathit{op}\ e_2) = & \text{let} & (\widehat{\tau}_1, \theta_1, C_1) = \mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e_1) \\
 & & (\widehat{\tau}_2, \theta_2, C_2) = \mathcal{W}_{\mathsf{CFA}}(\theta_1\ \widehat{\Gamma}, e_2) \\
 & & \theta_3 = \mathcal{U}_{\mathsf{CFA}}(\theta_2\ \widehat{\tau}_1, \tau_{op}^1) \\
 & & \theta_4 = \mathcal{U}_{\mathsf{CFA}}(\theta_3\ \widehat{\tau}_2, \tau_{op}^2) \\
 & \text{in} & (\tau_{op}, \quad \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1, \\
 & & \quad \theta_4\ (\theta_3\ (\theta_2\ C_1)) \cup \theta_4\ (\theta_3\ C_2))
\end{array}
$$

Example:

$\mathcal{W}_{\mathsf{CFA}}([\,], (\mathtt{fn}_{\mathsf{X}}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{x})\ (\mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y}))$

call $\mathcal{W}_{\mathsf{CFA}}([\,], \mathtt{fn}_{\mathsf{X}}\ \mathtt{x}\ \mathtt{=>}\ \mathtt{x})$

create the fresh type variable $'\mathtt{a}$ and the annotation variable $'1$

return $('\mathtt{a} \xrightarrow{'1} '\mathtt{a}, \mathit{id}, \{'1 \supseteq \{\mathsf{X}\}\})$

call $\mathcal{W}_{\mathsf{CFA}}([\,], \mathtt{fn}_{\mathsf{Y}}\ \mathtt{y}\ \mathtt{=>}\ \mathtt{y})$

create the fresh type variable $'\mathtt{b}$ and the annotation variable $'2$

return $('\mathtt{b} \xrightarrow{'2} '\mathtt{b}, \mathit{id}, \{'2 \supseteq \{\mathsf{Y}\}\})$

create the fresh type variable $'\mathtt{c}$ and the annotation variable $'3$

call $\mathcal{U}_{\mathsf{CFA}}('\mathtt{a} \xrightarrow{'1} '\mathtt{a},\ ('\mathtt{b} \xrightarrow{'2} '\mathtt{b}) \xrightarrow{'3} '\mathtt{c})$

return $['3 \mapsto '1]['\mathtt{a} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]['\mathtt{c} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]$

return $('\mathtt{b} \xrightarrow{'2} '\mathtt{b}, ['3 \mapsto '1]['\mathtt{a} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}]['\mathtt{c} \mapsto '\mathtt{b} \xrightarrow{'2} '\mathtt{b}], \{'1 \supseteq \{\mathsf{X}\}, '2 \supseteq \{\mathsf{Y}\}\})$

Example:

$$\mathcal{W}_{\mathsf{CFA}}([\,],\ \begin{array}{l}\texttt{let g = (fun}_{\mathsf{F}}\texttt{ f x => f (fn}_{\mathsf{Y}}\texttt{ y => y))} \\ \texttt{in g (fn}_{\mathsf{Z}}\texttt{ z => z))}\end{array}$$

$$= ('a, \cdots, \{'1 \supseteq \{\mathsf{F}\}, '3 \supseteq \{\mathsf{Y}\}, '3 \supseteq \{\mathsf{Z}\}\})$$

Syntactic soundness theorem

If $\boxed{\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e) = (\widehat{\tau}, \theta, C)}$ and θ_G is a *ground validation* of $\theta\ \widehat{\Gamma}$, $\widehat{\tau}$ and C
then $\boxed{\theta_G(\theta\ \widehat{\Gamma})\ \vdash_{\mathsf{CFA}}\ e : \theta_G\ \widehat{\tau}}$

θ_G is a ground validation of $\widehat{\Gamma}'$, $\widehat{\tau}$ and C if and only if

- θ_G is defined on all type and annotation variables in $\widehat{\Gamma}'$, $\widehat{\tau}$ and C
- θ_G maps all type and annotation variables in its domain to types and annotations without variables
- θ_G is a solution to the constraints of C: $\theta_G \models C$

Question: What happens if C does not have a solution?

Syntactic completeness theorem

Assume that $\widehat{\Gamma}$ is a "simple" type environment and that $\theta' \ \widehat{\Gamma} \ \vdash_{\mathsf{CFA}} \ e : \widehat{\tau}'$ for some ground substitution θ'. Then there exists $\widehat{\tau}$, θ, C and θ_G such that

1. $\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e) = (\widehat{\tau}, \theta, C)$,
2. θ_G is a ground validation of $\theta \ \widehat{\Gamma}$, $\widehat{\tau}$ and C,
3. $\theta_G \circ \theta = \theta'$ except on fresh type and annotation variables (as created by $\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e)$), and
4. $\theta_G \ \widehat{\tau} = \widehat{\tau}'$

The soundness result together with (1) and (2) gives $\boxed{\theta_G(\theta \ \widehat{\Gamma}) \vdash_{\mathsf{CFA}} e : \theta_G \ \widehat{\tau}}$

and by (3) and (4) this is equivalent to $\boxed{\theta' \ \widehat{\Gamma} \vdash_{\mathsf{CFA}} e : \widehat{\tau}'}$

The syntactic soundness theorem revisited

Problem: If the constraints generated by $\mathcal{W}_{\mathsf{CFA}}$ cannot be solved then we cannot use the soundness result to guarantee that the result produced by $\mathcal{W}_{\mathsf{CFA}}$ can be inferred in the inference system.

But the constraints always have solutions:

Lemma: If $\mathcal{W}_{\mathsf{CFA}}(\widehat{\Gamma}, e) = (\widehat{\tau}, \theta, C)$ and X is the set of annotation variables in C then

$$\{\theta_A \mid \theta_A \models C \wedge \mathit{dom}(\theta_A) = X\}$$

is a Moore family.

The least substitution solving C turns out to be

$$\theta_A\ \beta = \begin{cases} \{\pi \mid \beta \supseteq \{\pi\} \text{ is in } C\} & \text{if } \beta \text{ is in } C \\ \text{undefined} & \text{otherwise} \end{cases}$$

Side Effect Analysis

The language: an extension of Fun with imperative constructs for creating reference variables and for accessing and updating their values:

$$e ::= \cdots \mid \texttt{new}_\pi\ r\ \texttt{:=}\ e_1\ \texttt{in}\ e_2 \mid \texttt{!}r \mid r\ \texttt{:=}\ e_0 \mid e_1\ \texttt{;}\ e_2$$

Example:

$$\begin{array}{l}
\texttt{new}_{\mathsf{R}}\ \texttt{r := 0} \\
\texttt{in}\quad \texttt{let fib} = \texttt{fun f z => if z<3 then r:=!r+1} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{else f(z-1); f(z-2)} \\
\qquad\ \ \texttt{in}\ \ \texttt{fib x; !r}
\end{array}$$

Analysis result: $\texttt{fib} : \texttt{int} \xrightarrow{\{\texttt{!}\mathsf{R},\mathsf{R}\texttt{:=}\}} \texttt{int}$

Semantics (1)

We introduce *locations* $\xi \in \mathbf{Loc}$ in order to distinguish between the various incarnations of the `new`-construct – the configurations will then contain a *store* component

$$\varsigma \in \mathbf{Store} = \mathbf{Loc} \to_{\mathsf{fin}} \mathbf{Val}$$

where $v \in \mathbf{Val}$ is given by

$$v ::= c \mid \mathtt{fn}\ x\ \texttt{=>}\ e \mid \xi \qquad \text{(closed expressions only)}$$

Semantics (2)

$$\frac{\vdash \langle e_1, \varsigma_1 \rangle \longrightarrow \langle v_1, \varsigma_2 \rangle \qquad \vdash \langle e_2[r \mapsto \xi], \varsigma_2[\xi \mapsto v_1] \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}{\vdash \langle \mathtt{new}_\pi\ r\ \mathtt{:=}\ e_1\ \mathtt{in}\ e_2, \varsigma_1 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}$$

where ξ does not occur in the domain of ς_2

$$\vdash \langle \mathtt{!}\xi, \varsigma \rangle \longrightarrow \langle \varsigma(\xi), \varsigma \rangle$$

$$\frac{\vdash \langle e, \varsigma_1 \rangle \longrightarrow \langle v, \varsigma_2 \rangle}{\vdash \langle \xi \mathtt{:=} e, \varsigma_1 \rangle \longrightarrow \langle v, \varsigma_2[\xi \mapsto v] \rangle}$$

$$\frac{\vdash \langle e_1, \varsigma_1 \rangle \longrightarrow \langle v_1, \varsigma_2 \rangle \qquad \vdash \langle e_2, \varsigma_2 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}{\vdash \langle e_1; e_2, \varsigma_1 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}$$

Side Effect Analysis

$$\widehat{\Gamma} \vdash_{SE} e : \widehat{\tau} \;\&\; \boxed{\varphi}$$

$$\varphi ::= \{\texttt{!}\pi\} \mid \{\pi\texttt{:=}\} \mid \{\texttt{new}\,\pi\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset$$

$$\widehat{\tau} ::= \texttt{int} \mid \texttt{bool} \mid \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \mid \texttt{ref}_\pi\ \widehat{\tau}$$

$$\widehat{\Gamma} ::= [\,] \mid \widehat{\Gamma}[x \mapsto \widehat{\tau}]$$

Example:
```
new_R r := 0
in    let fib = fun f z => if z<3 then r:=!r+1
                                  else f(z-1); f(z-2)
      in  fib x; !r
```

$$[\texttt{x} \mapsto \texttt{int}][\texttt{r} \mapsto \texttt{ref}_{R}\ \texttt{int}] \vdash_{SE} \texttt{fib} : \texttt{int} \xrightarrow{\{\texttt{!}R, R\texttt{:=}\}} \texttt{int} \;\&\; \emptyset$$

$$[\cdots][\texttt{r} \mapsto \texttt{ref}_{R}\ \texttt{int}] \vdash_{SE} \texttt{r:=!r+1} : \texttt{int} \;\&\; \{\texttt{!}R, R\texttt{:=}\}$$

Side Effect Analysis (1)

$$\widehat{\Gamma} \vdash_{\mathsf{SE}} c : \tau_c \ \& \ \emptyset$$

$$\widehat{\Gamma} \vdash_{\mathsf{SE}} x : \widehat{\tau} \ \& \ \emptyset \qquad \text{if } \widehat{\Gamma}(x) = \widehat{\tau}$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{SE}} e_0 : \widehat{\tau}_0 \ \& \ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \texttt{fn}\ x\ \texttt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \emptyset}$$

$$\frac{\widehat{\Gamma}[f \mapsto \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0][x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{SE}} e_0 : \widehat{\tau}_0 \ \& \ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \texttt{fun}\ f\ x\ \texttt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \emptyset}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_2 \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1\ e_2 : \widehat{\tau}_0 \ \& \ \varphi_1 \cup \varphi_2 \cup \varphi_0}$$

Side Effect Analysis (2)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_0 : \texttt{bool} \;\&\; \varphi_0 \qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau} \;\&\; \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_2 : \widehat{\tau} \;\&\; \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 : \widehat{\tau} \;\&\; \varphi_0 \cup \varphi_1 \cup \varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_1 \;\&\; \varphi_1 \qquad \widehat{\Gamma}[x \mapsto \widehat{\tau}_1] \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \;\&\; \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \widehat{\tau}_2 \;\&\; \varphi_1 \cup \varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \tau_{op}^1 \;\&\; \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_2 : \tau_{op}^2 \;\&\; \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 \; op \; e_2 : \tau_{op} \;\&\; \varphi_1 \cup \varphi_2}$$

Side Effect Analysis (3)

$$\widehat{\Gamma} \vdash_{\mathsf{SE}} \mathtt{!}x : \widehat{\tau} \ \& \ \{\mathtt{!}\pi\} \qquad \text{if } \widehat{\Gamma}(x) = \mathtt{ref}_\pi \widehat{\tau}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau} \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{SE}} x \ \mathtt{:=}\ e : \widehat{\tau} \ \& \ \varphi \cup \{\pi\mathtt{:=}\}} \qquad \text{if } \widehat{\Gamma}(x) = \ \mathtt{ref}_\pi \widehat{\tau}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_1 \ \& \ \varphi_1 \qquad \widehat{\Gamma}[x \mapsto \mathtt{ref}_\pi \widehat{\tau}_1] \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \mathtt{new}_\pi \ x \ \mathtt{:=}\ e_1 \ \mathtt{in}\ e_2 : \widehat{\tau}_2 \ \& \ (\varphi_1 \cup \varphi_2 \cup \{\mathtt{new}\,\pi\})}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_1 \ \& \ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 \ \mathtt{;}\ e_2 : \widehat{\tau}_2 \ \& \ \varphi_1 \cup \varphi_2}$$

Example:

$\texttt{int}\ \&\ \{\texttt{new}B, \texttt{!}A, A\texttt{:=}, \texttt{!}B\}$

$$
\begin{array}{ll}
\texttt{new}_A\ \texttt{x:=1} & \\
\texttt{in} & \boxed{\texttt{(new}_B\ \texttt{y:=!x in (x:=!y+1; !y+3))}} \\
 & \\
\quad\ \texttt{+} & \boxed{\texttt{(new}_C\ \texttt{x:=!x in (x:=!x+1; !x+1))}}
\end{array}
$$

$\texttt{int}\ \&\ \{\texttt{new}C, \texttt{!}A, C\texttt{:=}, \texttt{!}C\}$

For the overall program:

$$\texttt{int}\ \&\ \{\texttt{new}A, A\texttt{:=}, \texttt{!}A, \texttt{new}B, \texttt{!}B, \texttt{new}C, C\texttt{:=}, \texttt{!}C\}$$

Subeffecting and subtyping

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau} \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau}' \ \& \ \varphi'} \quad \text{if } \boxed{\widehat{\tau} \leq \widehat{\tau}'} \text{ and } \boxed{\varphi \subseteq \varphi'}$$

subeffecting

subtyping

$\varphi \subseteq \varphi'$ means that φ is "a subset" of φ'

$\widehat{\tau} \leq \widehat{\tau}'$ is defined by

shape conformant subtyping

$$\widehat{\tau} \leq \widehat{\tau} \qquad \frac{\widehat{\tau}_1' \leq \widehat{\tau}_1 \quad \varphi \subseteq \varphi' \quad \widehat{\tau}_2 \leq \widehat{\tau}_2'}{\widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \leq \widehat{\tau}_1' \xrightarrow{\varphi'} \widehat{\tau}_2'} \qquad \frac{\widehat{\tau} \leq \widehat{\tau}' \quad \widehat{\tau}' \leq \widehat{\tau}}{\mathbf{ref}_\pi\ \widehat{\tau} \leq \mathbf{ref}_\pi\ \widehat{\tau}'}$$

The ordering on $\widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2$ is *contravariant* in $\widehat{\tau}_1$ and *covariant* in $\widehat{\tau}_2$

Example: subtyping

$$\texttt{int} \xrightarrow{\{!\mathsf{A}\}} \texttt{int}\ \&\ \emptyset \qquad\qquad \texttt{int} \xrightarrow{\{\mathsf{A}:=\}} \texttt{int}\ \&\ \emptyset$$

$$\begin{array}{l} \texttt{new}_{\mathsf{A}}\ \texttt{x:=1} \\ \texttt{in}\ \ \texttt{(fn f => f}\ \boxed{\texttt{(fn y => !x)}}\ \texttt{+ f}\ \boxed{\texttt{(fn z => (x:=z; z))}}\texttt{)} \\ \qquad \texttt{(fn g => g 1)} \end{array}$$

$$(\texttt{int} \xrightarrow{\{!\mathsf{A},\mathsf{A}:=\}} \texttt{int}) \xrightarrow{\{!\mathsf{A},\mathsf{A}:=\}} \texttt{int}$$

Subtyping:

$$\texttt{int} \xrightarrow{\{!\mathsf{A}\}} \texttt{int} \ \leq\ \texttt{int} \xrightarrow{\{!\mathsf{A},\mathsf{A}:=\}} \texttt{int}$$

$$\texttt{int} \xrightarrow{\{\mathsf{A}:=\}} \texttt{int} \ \leq\ \texttt{int} \xrightarrow{\{!\mathsf{A},\mathsf{A}:=\}} \texttt{int}$$

Example: subeffecting

$$\texttt{int} \xrightarrow{\{!\mathsf{A},\ \mathsf{A}{:=}\}} \texttt{int}\ \&\ \emptyset \qquad\qquad \texttt{int} \xrightarrow{\{!\mathsf{A},\mathsf{A}{:=}\}} \texttt{int}\ \&\ \emptyset$$

$\texttt{new}_{\mathsf{A}}$ `x:=1`

```
in  (fn f => f (fn y => !x) + f (fn z => (x:=z; z)))
    (fn g => g 1)
```

$$(\texttt{int} \xrightarrow{\{!\mathsf{A},\mathsf{A}{:=}\}} \texttt{int}) \xrightarrow{\{!\mathsf{A},\mathsf{A}{:=}\}} \texttt{int}$$

Exception Analysis

The language: an extension of Fun with constructs for raising and handling exceptions:

$$e ::= \cdots \mid \texttt{raise}\ s \mid \texttt{handle}\ s\ \texttt{as}\ e_1\ \texttt{in}\ e_2$$

where s is a string (a constant)

Example:

```
handle pos as z := 1000
in let f = fn g => fn x => g x
   in   f (fn y => if y < 0 then raise neg else y) (3-2)
      + f (fn z => if z > 0 then raise pos else 0-z) (2-3)
```

Analysis result for the first argument to f: $\texttt{int} \xrightarrow{\{\texttt{neg}\}} \texttt{int}\ \&\ \emptyset$

the first argument to f: $\texttt{int} \xrightarrow{\{\texttt{pos}\}} \texttt{int}\ \&\ \emptyset$

the whole program: $\texttt{int}\ \&\ \{\texttt{neg}\}$

Semantics (1)

Values $v \in \mathbf{Val}$ can be raised exceptions:

$$v ::= c \mid \texttt{fn x => } e \mid \texttt{raise } s \qquad \text{(closed expressions only)}$$

The semantics of the new constructs:

$$\vdash \texttt{raise } s \longrightarrow \texttt{raise } s$$

$$\frac{\vdash e_2 \longrightarrow v_2}{\vdash \texttt{handle } s \texttt{ as } e_1 \texttt{ in } e_2 \longrightarrow v_2} \quad \text{if } v_2 \neq \texttt{raise } s$$

$$\frac{\vdash e_2 \longrightarrow \texttt{raise } s \qquad \vdash e_1 \longrightarrow v_1}{\vdash \texttt{handle } s \texttt{ as } e_1 \texttt{ in } e_2 \longrightarrow v_1}$$

Semantics (2)

New rules for the old constructs:

$$\frac{\vdash e_1 \longrightarrow \texttt{raise}\ s}{\vdash e_1\ e_2 \longrightarrow \texttt{raise}\ s}$$

$$\frac{\vdash e_1 \longrightarrow (\texttt{fn}\ x\ \texttt{=>}\ e_0) \quad \vdash e_2 \rightarrow \texttt{raise}\ s}{\vdash e_1\ e_2 \longrightarrow \texttt{raise}\ s}$$

$$\frac{\vdash e_1 \longrightarrow (\texttt{fn}\ x\ \texttt{=>}\ e_0) \quad \vdash e_2 \rightarrow v_2 \quad \vdash e_0[x \mapsto v_2] \longrightarrow \texttt{raise}\ s}{\vdash e_1\ e_2 \longrightarrow \texttt{raise}\ s}$$

plus similar rules for the other constructs

Exception Analysis

$$\widehat{\Gamma} \vdash_{ES} e : \widehat{\sigma}\ \&\ \varphi$$

polymorphism

$$\varphi ::= \{s\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset \mid \beta$$

$$\widehat{\sigma} ::= \forall(\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_m).\widehat{\tau}$$

$$\widehat{\tau} ::= \texttt{int} \mid \texttt{bool} \mid \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \mid \alpha$$

$$\widehat{\Gamma} ::= [\ ] \mid \widehat{\Gamma}[x \mapsto \widehat{\tau}]$$

Example:

```
handle pos as z := 1000
in let f = fn g => fn x => g x
   in   f (fn y => if y < 0 then raise neg else y) (3-2)
      + f (fn z => if z > 0 then raise pos else 0-z) (2-3)
```

Typing judgement:

$$[\ ] \vdash_{ES} \texttt{fn g => fn x => g x} : \forall\ \text{'a}, \text{'b}, \text{'1}.\ (\text{'a} \xrightarrow{\text{'1}} \text{'b}) \xrightarrow{\emptyset} (\text{'a} \xrightarrow{\text{'1}} \text{'b})\ \&\ \emptyset$$

Exception Analysis (1)

$$\widehat{\Gamma} \vdash_{\mathsf{ES}} c : \tau_c \;\&\; \emptyset$$

$$\widehat{\Gamma} \vdash_{\mathsf{ES}} x : \widehat{\sigma} \;\&\; \emptyset \qquad \text{if } \widehat{\Gamma}(x) = \widehat{\sigma}$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{ES}} e_0 : \widehat{\tau}_0 \;\&\; \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{ES}} \mathtt{fn}\ x\ \mathtt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \;\&\; \emptyset}$$

$$\frac{\widehat{\Gamma}[f \mapsto \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0][x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{ES}} e_0 : \widehat{\tau}_0 \;\&\; \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{ES}} \mathtt{fun}\ f\ x\ \mathtt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \;\&\; \emptyset}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 : \widehat{\tau}_2 \xrightarrow{\varphi_0} \widehat{\tau}_0 \;\&\; \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{ES}} e_2 : \widehat{\tau}_2 \;\&\; \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1\ e_2 : \widehat{\tau}_0 \;\&\; \varphi_1 \cup \varphi_2 \cup \varphi_0}$$

Exception Analysis (2)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_0 : \texttt{bool} \ \& \ \varphi_0 \quad \widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 : \widehat{\tau} \ \& \ \varphi_1 \quad \widehat{\Gamma} \vdash_{\mathsf{ES}} e_2 : \widehat{\tau} \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{ES}} \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 : \widehat{\tau} \ \& \ \varphi_0 \cup \varphi_1 \cup \varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 : \widehat{\sigma}_1 \ \& \ \varphi_1 \quad \widehat{\Gamma}[x \mapsto \widehat{\sigma}_1] \vdash_{\mathsf{ES}} e_2 : \widehat{\tau}_2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{ES}} \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \widehat{\tau}_2 \ \& \ \varphi_1 \cup \varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 : \tau_{op}^1 \ \& \ \varphi_1 \quad \widehat{\Gamma} \vdash_{\mathsf{ES}} e_2 : \tau_{op}^2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 \ op \ e_2 : \tau_{op} \ \& \ \varphi_1 \cup \varphi_2}$$

Exception Analysis (3)

$$\widehat{\Gamma} \vdash_{\mathsf{ES}} \texttt{raise}\ s : \widehat{\tau}\ \&\ \{s\}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e_1 : \widehat{\tau}\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{ES}} e_2 : \widehat{\tau}\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{ES}} \texttt{handle}\ s\ \texttt{as}\ e_1\ \texttt{in}\ e_2 : \widehat{\tau}\ \&\ \underbrace{\varphi_1 \cup (\varphi_2 \backslash \{s\})}_{\varphi_1 \text{ only needed if } s \in \varphi_2}}$$

Recall: $\varphi ::= \{s\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset \mid \beta$

$$\begin{array}{rcl}
\{s'\}\backslash\{s\} & = & \begin{cases} \emptyset & \text{if } s = s' \\ \{s'\} & \text{otherwise} \end{cases} \\
(\varphi \cup \varphi')\backslash\{s\} & = & (\varphi\backslash\{s\}) \cup (\varphi'\backslash\{s\}) \\
\emptyset\backslash\{s\} & = & \emptyset \\
\beta\backslash\{s\} & = & \beta \qquad \text{(the best we can do)}
\end{array}$$

Alternative: take $\varphi ::= \cdots \mid \varphi \backslash \{s\}$ and axiomatise set difference

Exception Analysis (4)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : \widehat{\tau} \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : \widehat{\tau}' \ \& \ \varphi'}$$

if $\widehat{\tau} \ \leq \ \widehat{\tau}'$ and $\varphi \ \subseteq \ \varphi'$

$\boxed{\text{shape conformant subtyping}}$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : \widehat{\tau} \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : \forall(\alpha_1, \cdots, \beta_1, \cdots).\ \widehat{\tau} \ \& \ \varphi}$$

if $\alpha_1, \cdots, \beta_1, \cdots$
do not occur free in $\widehat{\Gamma}$ and φ

$\boxed{\text{generalisation}}$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : \forall(\alpha_1, \cdots, \beta_1, \cdots).\ \widehat{\tau} \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{ES}} e : (\theta \ \widehat{\tau}) \ \& \ \varphi}$$

if θ has $\mathit{dom}(\theta) \subseteq \{\alpha_1, \cdots, \beta_1, \cdots\}$

$\boxed{\text{instantiation}}$

Example: polymorphism

$$\mathtt{int} \xrightarrow{\{\mathtt{neg}\}} \mathtt{int}\ \&\ \emptyset$$

```
handle pos as z := 1000
in let f = fn g => fn x => g x
   in  f (fn y => if y < 0 then raise neg else y)  (3-2)
     + f (fn z => if z > 0 then raise pos else 0-z)  (2-3)
```

$$\forall\ 'a,'b,'1.\ ('a \xrightarrow{'1} 'b) \xrightarrow{\emptyset} ('a \xrightarrow{'1} 'b) \qquad\qquad \mathtt{int} \xrightarrow{\{\mathtt{pos}\}} \mathtt{int}\ \&\ \emptyset$$

Instantiations:

$$\begin{aligned} \mathtt{f}:\quad & ['a \mapsto \mathtt{int}; 'b \mapsto \mathtt{int}; '1 \mapsto \{\mathtt{neg}\}](('a \xrightarrow{'1} 'b) \xrightarrow{\emptyset} ('a \xrightarrow{'1} 'b)) \\ & = (\mathtt{int} \xrightarrow{\{\mathtt{neg}\}} \mathtt{int}) \xrightarrow{\emptyset} (\mathtt{int} \xrightarrow{\{\mathtt{neg}\}} \mathtt{int}) \end{aligned}$$

$$\begin{aligned} \mathtt{f}:\quad & ['a \mapsto \mathtt{int}; 'b \mapsto \mathtt{int}; '1 \mapsto \{\mathtt{pos}\}](('a \xrightarrow{'1} 'b) \xrightarrow{\emptyset} ('a \xrightarrow{'1} 'b)) \\ & = (\mathtt{int} \xrightarrow{\{\mathtt{pos}\}} \mathtt{int}) \xrightarrow{\emptyset} (\mathtt{int} \xrightarrow{\{\mathtt{pos}\}} \mathtt{int}) \end{aligned}$$

Example: subtyping

$$\mathtt{int} \xrightarrow{\{\mathrm{neg}\}} \mathtt{int}\ \&\ \emptyset$$

```
handle pos as z := 1000
in let f = fn g => fn x => g x
    in  f (fn y => if y < 0 then raise neg else y) (3-2)
      + f (fn z => if z > 0 then raise pos else 0-z) (2-3)
```

$$(\mathtt{int} \xrightarrow{\{\mathrm{neg,pos}\}} \mathtt{int}) \xrightarrow{\emptyset} (\mathtt{int} \xrightarrow{\{\mathrm{neg,pos}\}} \mathtt{int}) \qquad \mathtt{int} \xrightarrow{\{\mathrm{pos}\}} \mathtt{int}\ \&\ \emptyset$$

Subtyping:

$$\mathtt{int} \xrightarrow{\{\mathrm{neg}\}} \mathtt{int} \ \le\ \mathtt{int} \xrightarrow{\{\mathrm{neg,pos}\}} \mathtt{int}$$

$$\mathtt{int} \xrightarrow{\{\mathrm{pos}\}} \mathtt{int} \ \le\ \mathtt{int} \xrightarrow{\{\mathrm{neg,pos}\}} \mathtt{int}$$

Example: subeffecting

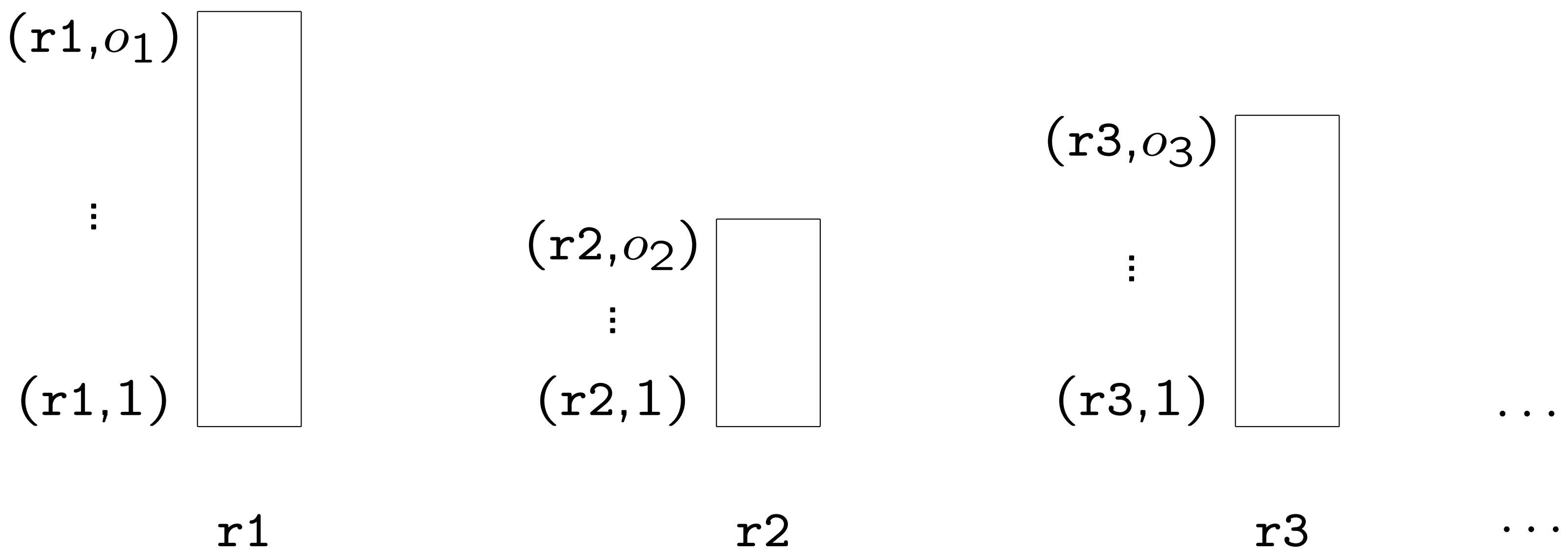
$$\texttt{int} \xrightarrow{\{\texttt{neg},\, \texttt{pos}\}} \texttt{int}\ \&\ \emptyset$$

```
handle pos as z := 1000
in let f = fn g => fn x => g x
    in   f (fn y => if y < 0 then raise neg else y)   (3-2)
       + f (fn z => if z > 0 then raise pos else 0-z) (2-3)
```

$$(\texttt{int} \xrightarrow{\{\texttt{neg},\texttt{pos}\}} \texttt{int}) \xrightarrow{\emptyset} (\texttt{int} \xrightarrow{\{\texttt{neg},\texttt{pos}\}} \texttt{int}) \qquad \texttt{int} \xrightarrow{\{\texttt{neg},\, \texttt{pos}\}} \texttt{int}\ \&\ \emptyset$$

Region Inference

Memory model for stack-based implementation of Fun



Region inference: determines how far locally allocated data can be passed around and when the allocated space can be reclaimed

Region Inference

The language: an extension of Fun with explicit region information:

$$
\begin{array}{rcl}
ee & ::= & c \ \texttt{at}\ r \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ ee_0\ \texttt{at}\ r \mid \texttt{fun}\ f\ [\vec{\varrho}]\ x\ \texttt{=>}\ ee_0\ \texttt{at}\ r \mid ee_1\ ee_2 \\
& \mid & \texttt{if}\ ee_0\ \texttt{then}\ ee_1\ \texttt{else}\ ee_2 \mid \texttt{let}\ x\ \texttt{=}\ ee_1\ \texttt{in}\ ee_2 \mid ee_1\ op\ ee_2\ \texttt{at}\ r \\
& \mid & \underbrace{ee[\vec{r}]\ \texttt{at}\ r}_{\text{copy}} \mid \underbrace{\texttt{letregion}\ \vec{\varrho}\ \texttt{in}\ ee}_{\text{local region}}
\end{array}
$$

where

$$
\begin{array}{rcll}
rn & ::= & \mathrm{r1} \mid \mathrm{r2} \mid \mathrm{r3} \mid \cdots & \text{region names} \\
\varrho & ::= & ''1 \mid ''2 \mid ''3 \mid \cdots & \text{region variables} \\
r & ::= & \varrho \mid rn & \text{regions}
\end{array}
$$

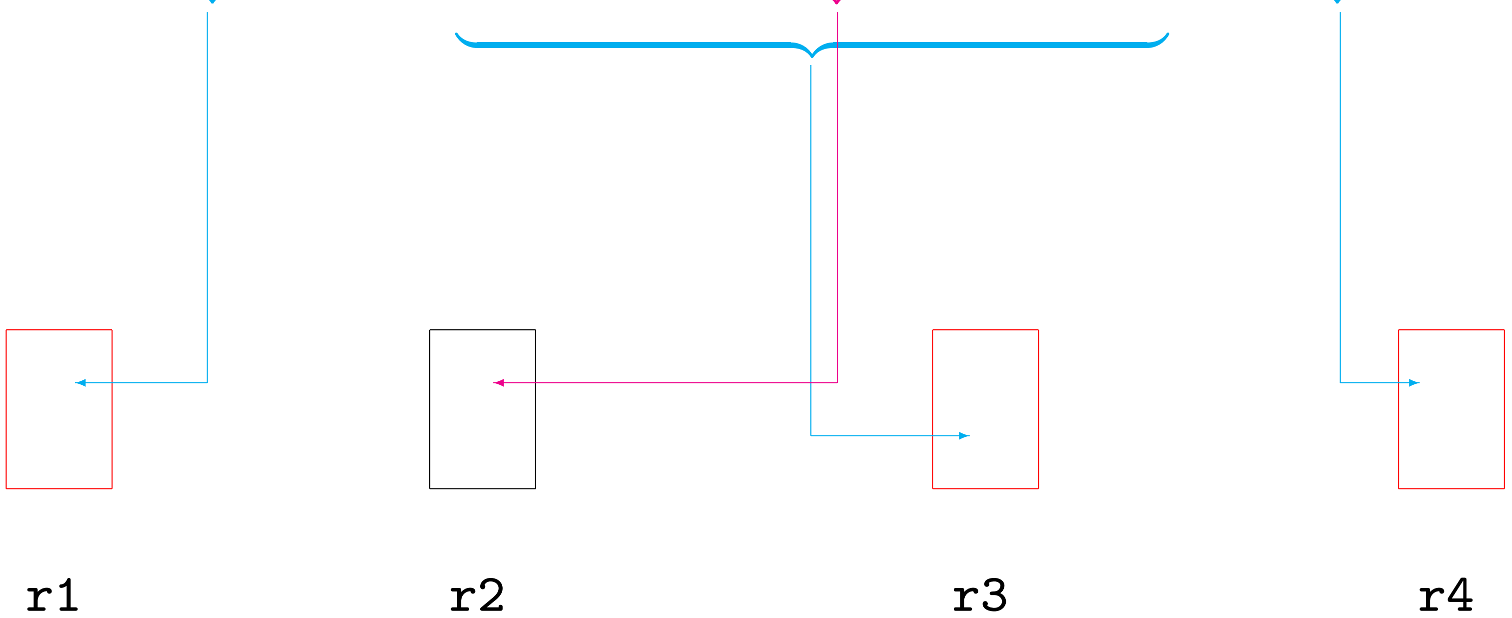
Example:

Expression

(let x = 7 in (fn y => y+x)) 9

Extended expression

letregion ϱ_1, ϱ_3, ϱ_4
in (let x = (7 at ϱ_1) in (fn y => (y+x at ϱ_2) at ϱ_3)) (9 at ϱ_4)



Semantics

$$\rho \vdash \langle ee, \varsigma \rangle \longrightarrow \langle {\color{red}v}, \varsigma' \rangle$$

store: $\mathbf{Store} = \mathbf{RName} \to_{\mathsf{fin}} (\mathbf{Offset} \to_{\mathsf{fin}} {\color{red}\mathbf{SVal}})$

value: ${\color{red}v = (rn, o)} \in \mathbf{RName} \times \mathbf{Offset}$

environment: $\rho \in \mathbf{Env} = \mathbf{Var}_\star \to \mathbf{RName} \times \mathbf{Offset}$

Storable values ${\color{red}w} \in {\color{red}\mathbf{SVal}}$ are given by

$$w ::= c \mid \underbrace{\texttt{close fn } x \texttt{ => } ee \texttt{ in } \rho}_{\text{ordinary closure}} \mid \underbrace{\texttt{reg-close } [\vec{\varrho}] \texttt{ fn } x \texttt{ => } ee \texttt{ in } \rho}_{\text{region polymorphic closure}}$$

Semantics (1)

$$\rho \vdash \langle c \texttt{ at } rn, \varsigma\rangle \longrightarrow \langle (rn,o), \varsigma[(rn,o) \mapsto c]\rangle \qquad \text{if } o \notin dom(\varsigma(rn))$$

$$\rho \vdash \langle x, \varsigma\rangle \longrightarrow \langle \rho(x), \varsigma\rangle$$

$$\begin{array}{l}\rho \vdash \langle (\texttt{fn } x \texttt{ => } ee_0) \texttt{ at } rn, \varsigma\rangle \longrightarrow \\ \qquad \langle (rn,o), \varsigma[(rn,o) \mapsto \texttt{close fn } x \texttt{ => } ee_0 \texttt{ in } \rho]\rangle \qquad \text{if } o \notin dom(\varsigma(rn))\end{array}$$

$$\begin{array}{l}\rho \vdash \langle (\texttt{fun } f[\vec{\varrho}]\ x \texttt{ => } ee_0) \texttt{ at } rn, \varsigma\rangle \longrightarrow \\ \qquad \langle (rn,o), \varsigma[(rn,o) \mapsto \texttt{reg-close } [\vec{\varrho}] \texttt{ fn } x \texttt{ => } ee \texttt{ in } \rho[f \mapsto (rn,o)]]\rangle \\ \qquad\qquad \text{if } o \notin dom(\varsigma(rn))\end{array}$$

$$\frac{\rho \vdash \langle ee_1, \varsigma_1\rangle \longrightarrow \langle (rn_1,o_1), \varsigma_2\rangle \quad \rho \vdash \langle ee_2, \varsigma_2\rangle \longrightarrow \langle v_2, \varsigma_3\rangle \quad \rho_0[x \mapsto v_2] \vdash \langle ee_0, \varsigma_3\rangle \longrightarrow \langle v_0, \varsigma_4\rangle}{\rho \vdash \langle ee_1\ ee_2, \varsigma_1\rangle \longrightarrow \langle v_0, \varsigma_4\rangle}$$

$$\text{if } \varsigma_3((rn_1,o_1)) = \texttt{close fn } x \texttt{ => } ee_0 \texttt{ in } \rho_0$$

Semantics (2)

$$\frac{\rho \vdash \langle ee_0, \varsigma_1 \rangle \longrightarrow \langle (rn, o), \varsigma_2 \rangle \quad \rho \vdash \langle ee_1, \varsigma_2 \rangle \longrightarrow \langle v_1, \varsigma_3 \rangle}{\rho \vdash \langle \texttt{if } ee_0 \texttt{ then } ee_1 \texttt{ else } ee_2, \varsigma_1 \rangle \longrightarrow \langle v_1, \varsigma_3 \rangle} \quad \text{if } \varsigma_2((rn, o)) = \texttt{true}$$

$$\frac{\rho \vdash \langle ee_0, \varsigma_1 \rangle \longrightarrow \langle (rn, o), \varsigma_2 \rangle \quad \rho \vdash \langle ee_2, \varsigma_2 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}{\rho \vdash \langle \texttt{if } ee_0 \texttt{ then } ee_1 \texttt{ else } ee_2, \varsigma_1 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle} \quad \text{if } \varsigma_2((rn, o)) = \texttt{false}$$

$$\frac{\rho \vdash \langle ee_1, \varsigma_1 \rangle \longrightarrow \langle v_1, \varsigma_2 \rangle \quad \rho[x \mapsto v_1] \vdash \langle ee_2, \varsigma_2 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}{\rho \vdash \langle \texttt{let } x \texttt{ = } ee_1 \texttt{ in } ee_2, \varsigma_1 \rangle \longrightarrow \langle v_2, \varsigma_3 \rangle}$$

$$\frac{\rho \vdash \langle ee_1, \varsigma_1 \rangle \longrightarrow \langle (rn_1, o_1), \varsigma_2 \rangle \quad \rho \vdash \langle ee_2, \varsigma_2 \rangle \longrightarrow \langle (rn_2, o_2), \varsigma_3 \rangle}{\rho \vdash \langle (ee_1 \ op \ ee_2) \texttt{ at } rn, \varsigma_1 \rangle \longrightarrow \langle (rn, o), \varsigma_3[(rn, o) \mapsto w] \rangle}$$

$$\text{if } \varsigma_3((rn_1, o_1)) \ \textbf{op} \ \varsigma_3((rn_2, o_2)) = w \text{ and } o \notin \mathit{dom}(\varsigma_3(rn))$$

Semantics (3)

$$\frac{\rho \vdash \langle ee, \varsigma_1\rangle \longrightarrow \langle (rn', o'), \varsigma_2\rangle}{\begin{array}{l}\rho \vdash \langle ee[\vec{rn}]\ \texttt{at}\ rn, \varsigma_1\rangle \longrightarrow \\ \qquad\qquad \langle (rn, o), \varsigma_2[(rn, o) \mapsto \texttt{close fn}\ x\ \texttt{=>}\ ee_0[\vec{\varrho} \mapsto \vec{rn}]\ \texttt{in}\ \rho_0]\rangle\end{array}}$$

if $o \notin \mathit{dom}(\varsigma_2(rn))$ and $\varsigma_2((rn', o')) = \texttt{reg-close}\ [\vec{\varrho}]\ \texttt{fn}\ x\ \texttt{=>}\ ee_0\ \texttt{in}\ \rho_0$

$$\frac{\rho \vdash \langle ee[\vec{\varrho} \mapsto \vec{rn}], \varsigma_1[\vec{rn} \mapsto \vec{[\ ]}]\rangle \longrightarrow \langle v, \varsigma_2\rangle}{\rho \vdash \langle \texttt{letregion}\ \vec{\varrho}\ \texttt{in}\ ee, \varsigma_1\rangle \longrightarrow \langle v, \varsigma_2 \backslash\!\backslash \vec{rn}\rangle} \qquad \text{if } \{\vec{rn}\} \cap \mathit{dom}(\varsigma) = \emptyset$$

where

$$(\varsigma \backslash\!\backslash \vec{rn})(rn, o) = \begin{cases} \varsigma(rn, o) & \text{if } (rn, o) \in \mathit{dom}(\varsigma) \setminus \{\vec{rn}\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Region Inference

$$\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : \widehat{\sigma}\ @r\ \&\ \varphi$$

polymorphic recursion

$$\varphi ::= \{\mathsf{put}\ r\} \mid \{\mathsf{get}\ r\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset \mid \beta$$

$$\widehat{\sigma} ::= \forall(\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_m), [\varrho_1, \cdots, \varrho_k].\widehat{\tau}$$

$$\mid \forall(\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_m).\widehat{\tau}$$

$$\widehat{\tau} ::= \mathtt{int} \mid \mathtt{bool} \mid (\widehat{\tau}_1 @ r_1) \xrightarrow{\beta.\varphi} (\widehat{\tau}_2 @ r_2) \mid \alpha$$

$$\widehat{\Gamma} ::= [\ ] \mid \widehat{\Gamma}[x \mapsto \widehat{\tau}]$$

Example:

$$[\mathtt{x} \mapsto \mathtt{int}@\varrho_1] \vdash_{\mathsf{RI}} \mathtt{(fn\ y\ =>\ y+x)} \leadsto \mathtt{(fn\ y\ =>\ (y+x)\ at}\ \varrho_2\mathtt{)\ at}\ \varrho_3 :$$

$$((\mathtt{int}@\varrho_4) \xrightarrow{\beta.\varphi} (\mathtt{int}@\varrho_2))@\varrho_3\ \&\ \emptyset$$

where $\varphi = \{\mathsf{get}\ \varrho_4, \mathsf{get}\ \varrho_1, \mathsf{put}\ \varrho_2\}$

Region Inference (1)

$$\widehat{\Gamma} \vdash_{\mathsf{RI}} c \rightsquigarrow c \texttt{ at } r : (\tau_c @ r) \ \& \ \{\mathsf{put}\ r\}$$

$$\widehat{\Gamma} \vdash_{\mathsf{RI}} x \rightsquigarrow x : \widehat{\sigma} \ \& \ \emptyset \qquad \text{if } \widehat{\Gamma}(x) = \widehat{\sigma}$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x @ r_x] \vdash_{\mathsf{RI}} e_0 \rightsquigarrow ee_0 : (\widehat{\tau}_0 @ r_0) \ \& \ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{RI}} \texttt{fn } x \texttt{ => } e_0 \rightsquigarrow \texttt{fn } x \texttt{ => } ee_0 \texttt{ at } r : ((\widehat{\tau}_x @ r_x \xrightarrow{\beta.\varphi_0} \widehat{\tau}_0 @ r_0) @ r) \ \& \ \{\mathsf{put}\ r\}}$$

$$\frac{\widehat{\Gamma}[f \mapsto \forall \vec{\beta}[\vec{\varrho}].\widehat{\tau} @ r] \vdash_{\mathsf{RI}} \texttt{fn } x \texttt{ => } e_0 \rightsquigarrow \texttt{fn } x \texttt{ => } ee_0 \texttt{ at } r : (\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} \texttt{fun } f \ x \texttt{ => } e_0 \rightsquigarrow \texttt{fun } f \ [\vec{\varrho}] \ x \texttt{ => } ee_0 \texttt{ at } r : (\forall \vec{\beta}[\vec{\varrho}].\widehat{\tau} @ r) \ \& \ \varphi}$$

if $\vec{\beta}$ and $\vec{\varrho}$ do not occur free in $\widehat{\Gamma}$ and φ

$$\frac{\begin{array}{c}\widehat{\Gamma} \vdash_{\mathsf{RI}} e_1 \rightsquigarrow ee_1 : ((\widehat{\tau}_2 @ r_2 \xrightarrow{\beta_0.\varphi_0} \widehat{\tau}_0 @ r_0) @ r_1) \ \& \ \varphi_1 \\ \widehat{\Gamma} \vdash_{\mathsf{RI}} e_2 \rightsquigarrow ee_2 : (\widehat{\tau}_2 @ r_2) \ \& \ \varphi_2\end{array}}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e_1 \ e_2 \rightsquigarrow ee_1 \ ee_2 : (\widehat{\tau}_0 @ r_0) \ \& \ \varphi_1 \cup \varphi_2 \cup \varphi_0 \cup \beta_0 \cup \{\mathsf{get}\ r_1\}}$$

Region Inference (2)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e_0 \leadsto ee_0 : (\mathtt{bool}@r_0)\ \&\ \varphi_0 \qquad \widehat{\Gamma} \vdash_{\mathsf{RI}} e_1 \leadsto ee_1 : (\widehat{\tau}@r)\ \&\ \varphi_1 \quad \widehat{\Gamma} \vdash_{\mathsf{RI}} e_2 \leadsto ee_2 : (\widehat{\tau}@r)\ \&\ \varphi_2}{\begin{array}{l}\widehat{\Gamma} \vdash_{\mathsf{RI}} \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \leadsto \mathtt{if}\ ee_0\ \mathtt{then}\ ee_1\ \mathtt{else}\ ee_2 : \\ \qquad\qquad (\widehat{\tau}@r)\ \&\ \varphi_0 \cup \varphi_1 \cup \varphi_2 \cup \{\mathsf{get}\ r_0\}\end{array}}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e_1 \leadsto ee_1 : (\widehat{\sigma}_1@r_1)\ \&\ \varphi_1 \qquad \widehat{\Gamma}[x \mapsto \widehat{\sigma}_1@r_1] \vdash_{\mathsf{RI}} e_2 \leadsto ee_2 : (\widehat{\tau}_2@r_2)\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{RI}} \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 \leadsto \mathtt{let}\ x\ \mathtt{=}\ ee_1\ \mathtt{in}\ ee_2 : (\widehat{\tau}_2@r_2)\ \&\ \varphi_1 \cup \varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e_1 \leadsto ee_1 : (\tau^1_{op}@r_1)\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{RI}} e_2 \leadsto ee_2 : (\tau^2_{op}@r_2)\ \&\ \varphi_2}{\begin{array}{l}\widehat{\Gamma} \vdash_{\mathsf{RI}} e_1\ op\ e_2 \leadsto (ee_1\ op\ ee_2)\ \mathtt{at}\ r : \\ \qquad (\tau_{op}@r)\ \&\ \varphi_1 \cup \varphi_2 \cup \{\mathsf{get}\ r_1, \mathsf{get}\ r_2, \mathsf{put}\ r\}\end{array}}$$

Region Inference (3)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\widehat{\tau}' @ r) \ \& \ \varphi'}$$
if $\boxed{\widehat{\tau} \ \leq \ \widehat{\tau}'}$ and $\boxed{\varphi \ \subseteq \ \varphi'}$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : ((\forall \vec{\alpha}.\widehat{\tau}) @ r) \ \& \ \varphi}$$
if $\vec{\alpha}$ do not occur free in $\widehat{\Gamma}$ and φ

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\forall \vec{\beta}[\vec{\varrho}].\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\forall \vec{\alpha}\vec{\beta}[\vec{\varrho}].\widehat{\tau} @ r) \ \& \ \varphi}$$
if $\vec{\alpha}$ do not occur free in $\widehat{\Gamma}$ and φ

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\forall \vec{\alpha}\vec{\beta}.\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\theta \ \widehat{\tau} @ r) \ \& \ \varphi}$$
if $\mathit{dom}(\theta) \subseteq \{\vec{\alpha}, \vec{\beta}\}$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\forall \vec{\alpha}\vec{\beta}[\vec{\varrho}].\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee[\theta \vec{\varrho}] \ \mathtt{at} \ r' : (\theta \ \widehat{\tau} @ r') \ \& \ \varphi \cup \{\mathsf{get}\ r, \mathsf{put}\ r'\}}$$
if $\mathit{dom}(\theta) \subseteq \{\vec{\alpha}, \vec{\beta}, \vec{\varrho}\}$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto ee : (\widehat{\tau} @ r) \ \& \ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{RI}} e \leadsto \mathtt{letregion} \ \vec{\varrho} \ \mathtt{in} \ ee : (\widehat{\tau} @ r) \ \& \ \varphi'}$$
if $\varphi' = \mathit{Observe}(\widehat{\Gamma}, \widehat{\tau}, r)(\varphi)$ and $\vec{\varrho}$ occurs in φ but not in φ'

Observable effect

$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\varphi)$: the part of φ that is visible from the outside (i.e. from $\widehat{\Gamma}$, $\widehat{\tau}$ and r')

$$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\{\mathsf{put}\ r\}) = \begin{cases} \{\mathsf{put}\ r\} & \text{if } r \text{ occurs in } \widehat{\Gamma},\widehat{\tau}, \text{ or } r' \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\{\mathsf{get}\ r\}) = \begin{cases} \{\mathsf{get}\ r\} & \text{if } r \text{ occurs in } \widehat{\Gamma},\widehat{\tau}, \text{ or } r' \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\varphi_1 \cup \varphi_2) = \mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\varphi_1) \cup \mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\varphi_2)$$

$$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\emptyset) = \emptyset$$

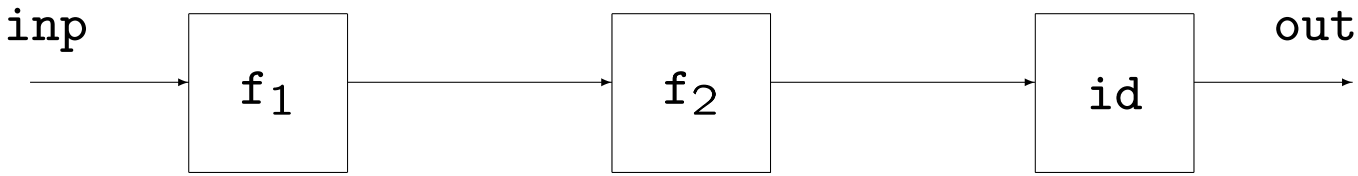
$$\mathit{Observe}(\widehat{\Gamma},\widehat{\tau},r')(\beta) = \begin{cases} \beta & \text{if } \beta \text{ occurs in } \widehat{\Gamma},\widehat{\tau}, \text{ or } r' \\ \emptyset & \text{otherwise} \end{cases}$$

Communication Analysis

The language: an extension of Fun with constructs for generating new processes, for communicating between processes over typed channels, and for creating new channels:

$$e ::= \cdots \mid \texttt{channel}_{\pi} \mid \texttt{spawn}\ e_0 \mid \texttt{send}\ e_1\ \texttt{on}\ e_2 \mid \texttt{receive}\ e_0 \mid e_1 ; e_2$$

Example: `pipe [f`$_1$`, f`$_2$`] inp out`



Example:

```
let node = fn_F f => fn_I inp => fn_O out =>
             spawn ((fun_H h d => let v = receive inp
                                  in send (f v) on out; h d) ())
in fun_P pipe fs => fn_I inp => fn_O out =>
        if isnil fs then node (fn_X x => x) inp out
        else let ch = channel_C
             in (node (hd fs) inp ch;
                 pipe (tl fs) ch out)
```

Behaviour for `node f in out`:

$$\mathsf{spawn}(\mathsf{rec}\ '0.\ (\underbrace{\texttt{in-chan?in-type}}_{\text{receive inp}}\ \boxed{;}\ \texttt{f-behaviour}\ \boxed{;}\ \underbrace{\texttt{out-chan!out-type}}_{\text{send}\ \cdots\ \text{on out}}\ \boxed{;}\ '0))$$

Sequential semantics

$$(\texttt{fn}_\pi\ x\ \texttt{=>}\ e)\ v \longrightarrow e[x \mapsto v]$$

$$\texttt{let}\ x\ \texttt{=}\ v\ \texttt{in}\ e \longrightarrow e[x \mapsto v]$$

$$v_1\ op\ v_2 \longrightarrow v \quad \text{if } v_1\ \mathbf{op}\ v_2 = v$$

$$\texttt{fun}_\pi\ f\ x\ \texttt{=>}\ e \longrightarrow (\texttt{fn}_\pi\ x\ \texttt{=>}\ e)[f \mapsto (\texttt{fun}_\pi\ f\ x\ \texttt{=>}\ e)]$$

$$\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_1$$

$$\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_2$$

$$v; e \longrightarrow e$$

Evaluation contexts:

$$\begin{array}{lcl} E & ::= & [\,]\ |\ E\ e\ |\ v\ E\ |\ \texttt{let}\ x\ \texttt{=}\ E\ \texttt{in}\ e\ |\ \texttt{if}\ E\ \texttt{then}\ e_1\ \texttt{else}\ e_2\ |\ E\ op\ e\ |\ v\ op\ E \\ & | & \texttt{send}\ E\ \texttt{on}\ e\ |\ \texttt{send}\ v\ \texttt{on}\ E\ |\ \texttt{receive}\ E\ |\ E; e \end{array}$$

Concurrent semantics

$$CP, PP[p : E[e_1]] \Rightarrow CP, PP[p : E[e_2]]$$

if $e_1 \rightarrow e_2$

$$CP, PP[p : E[\texttt{channel}_\pi]] \Rightarrow CP \cup \{ch\}, PP[p : E[ch]]$$

if $ch \notin CP$

$$CP, PP[p : E[\texttt{spawn}\ e_0]] \Rightarrow CP, PP[p : E[\texttt{()}]][p_0 : e_0]$$

if $p_0 \notin \mathit{dom}(PP) \cup \{p\}$

$$\begin{array}{l} CP, PP[p_1 : E_1[\texttt{send}\ v\ \texttt{on}\ ch]][p_2 : E_2[\texttt{receive}\ ch]] \\ \quad \Rightarrow CP, PP[p_1 : E_1[\texttt{()}]][p_2 : E_2[v]] \end{array}$$

if $p_1 \neq p_2$

Communication Analysis

$\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \widehat{\sigma} \ \& \ \varphi$

polymorphism & causality

$$\varphi ::= \Lambda \mid \varphi_1 ; \varphi_2 \mid \varphi_1 + \varphi_2 \mid \mathsf{rec}\beta.\varphi \mid \widehat{\tau}\ \mathsf{chan}\ r \mid \mathsf{spawn}\ \varphi \mid r!\widehat{\tau} \mid r?\widehat{\tau} \mid \beta$$

$$r ::= \{\pi\} \mid \emptyset \mid r_1 \cup r_2 \mid \varrho$$

$$\widehat{\tau} ::= \mathtt{int} \mid \mathtt{bool} \mid \mathtt{unit} \mid \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \mid \widehat{\tau}\ \mathsf{chan}\ r \mid \alpha$$

$$\widehat{\sigma} ::= \forall(\alpha_1, \cdots, \beta_1, \cdots, \varrho_1, \cdots).\widehat{\tau}$$

Example: `let node = fn`$_\mathsf{F}$ `f => fn`$_\mathsf{I}$ `inp => fn`$_\mathsf{O}$ `out =>`
`spawn ((fun`$_\mathsf{H}$ `h d => let v = receive inp`
`in send (f v) on out; h d) ())`

node: $\forall\ {'a}, {'b}, {'1}, {''1}, {''2}.\ \underbrace{({'a} \xrightarrow{'1} {'b})}_{\mathtt{f}} \xrightarrow{\Lambda} \underbrace{({'a}\ \mathtt{chan}\ {''1})}_{\mathtt{inp}} \xrightarrow{\Lambda} \underbrace{({'b}\ \mathtt{chan}\ {''2})}_{\mathtt{out}} \xrightarrow{\varphi} \mathtt{unit}$

where $\varphi = \mathsf{spawn}(\mathsf{rec}\ {'2}.\ ({''1}?{'a};\ {'1};\ {''2}!{'b};\ {'2}))$

Communication Analysis (1)

$$\widehat{\Gamma} \vdash_{\mathsf{CA}} c : \tau_c \ \& \ \Lambda$$

$$\widehat{\Gamma} \vdash_{\mathsf{CA}} x : \widehat{\sigma} \ \& \ \Lambda \qquad \text{if } \widehat{\Gamma}(x) = \widehat{\sigma}$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{CA}} e_0 : \widehat{\tau}_0 \ \& \ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{CA}} \mathtt{fn}_\pi \ x \ \texttt{=>} \ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \Lambda}$$

$$\frac{\widehat{\Gamma}[f \mapsto \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0][x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{CA}} e_0 : \widehat{\tau}_0 \ \& \ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{CA}} \mathtt{fun}_\pi \ f \ x \ \texttt{=>} \ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \Lambda}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_1 : \widehat{\tau}_2 \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \& \ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{CA}} e_2 : \widehat{\tau}_2 \ \& \ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_1 \ e_2 : \widehat{\tau}_0 \ \& \ \varphi_1 ; \varphi_2 ; \varphi_0}$$

Communication Analysis (2)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_0 : \mathtt{bool}\ \&\ \varphi_0 \qquad \widehat{\Gamma} \vdash_{\mathsf{CA}} e_1 : \widehat{\tau}\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{CA}} e_2 : \widehat{\tau}\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{CA}} \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \widehat{\tau}\ \&\ \varphi_0\,;(\varphi_1 + \varphi_2)}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_1 : \widehat{\sigma}_1\ \&\ \varphi_1 \qquad \widehat{\Gamma}[x \mapsto \widehat{\sigma}_1] \vdash_{\mathsf{CA}} e_2 : \widehat{\tau}_2\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{CA}} \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 : \widehat{\tau}_2\ \&\ \varphi_1\,;\varphi_2}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_1 : \tau_{op}^1\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathsf{CA}} e_2 : \tau_{op}^2\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{CA}} e_1\ op\ e_2 : \tau_{op}\ \&\ \varphi_1\,;\varphi_2\,;\Lambda}$$

Communication Analysis (3)

$$\widehat{\Gamma} \vdash_{CA} \texttt{channel}_\pi : \widehat{\tau}\ \mathsf{chan}\ \{\pi\}\ \&\ \widehat{\tau}\ \mathsf{chan}\ \{\pi\}$$

$$\frac{\widehat{\Gamma} \vdash_{CA} e_0 : \widehat{\tau}_0\ \&\ \varphi_0}{\widehat{\Gamma} \vdash_{CA} \texttt{spawn}\ e_0 : \texttt{unit}\ \&\ \mathsf{spawn}\ \varphi_0}$$

$$\frac{\widehat{\Gamma} \vdash_{CA} e_1 : \widehat{\tau}\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{CA} e_2 : \widehat{\tau}\ \texttt{chan}\ r_2\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{CA} \texttt{send}\ e_1\ \texttt{on}\ e_2 : \texttt{unit}\ \&\ \varphi_1 ; \varphi_2 ; r_2!\widehat{\tau}}$$

$$\frac{\widehat{\Gamma} \vdash_{CA} e_0 : \widehat{\tau}\ \texttt{chan}\ r_0\ \&\ \varphi_0}{\widehat{\Gamma} \vdash_{CA} \texttt{receive}\ e_0 : \widehat{\tau}\ \&\ \varphi_0 ; r_0?\widehat{\tau}}$$

$$\frac{\widehat{\Gamma} \vdash_{CA} e_1 : \widehat{\tau}_1\ \&\ \varphi_1 \qquad \widehat{\Gamma} \vdash_{CA} e_2 : \widehat{\tau}_2\ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{CA} e_1 ; e_2 : \tau_{op}\ \&\ \varphi_1 ; \varphi_2}$$

Communication Analysis (4)

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \widehat{\tau} \;\&\; \varphi}{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \widehat{\tau}' \;\&\; \varphi'}$$ if $\widehat{\tau} \leq \widehat{\tau}'$ and $\varphi \sqsubseteq \varphi'$

$$\widehat{\tau} \leq \widehat{\tau} \qquad \frac{\widehat{\tau}_1' \leq \widehat{\tau}_1 \quad \widehat{\tau}_2 \leq \widehat{\tau}_2' \quad \varphi \sqsubseteq \varphi'}{\widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2 \leq \widehat{\tau}_1' \xrightarrow{\varphi'} \widehat{\tau}_2'} \qquad \frac{\widehat{\tau} \leq \widehat{\tau}' \quad \widehat{\tau}' \leq \widehat{\tau} \quad r \subseteq r'}{\widehat{\tau}\ \mathtt{chan}\ r \leq \widehat{\tau}'\ \mathtt{chan}\ r'}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \widehat{\tau} \;\&\; \varphi}{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \forall(\alpha_1, \cdots, \beta_1, \cdots, \varrho_1, \cdots).\widehat{\tau} \;\&\; \varphi}$$ if $\alpha_1, \cdots, \beta_1, \cdots, \varrho_1, \cdots$ do not occur free in $\widehat{\Gamma}$ and φ

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : \forall(\alpha_1, \cdots, \beta_1, \cdots, \varrho_1, \cdots).\widehat{\tau} \;\&\; \varphi}{\widehat{\Gamma} \vdash_{\mathsf{CA}} e : (\theta\ \widehat{\tau}) \;\&\; \varphi}$$ if $dom(\theta) \subseteq \{\alpha_1, \cdots, \beta_1, \cdots, \varrho_1, \cdots\}$

Ordering on behaviours

$$\varphi \sqsubseteq \varphi \qquad \frac{\varphi_1 \sqsubseteq \varphi_2 \quad \varphi_2 \sqsubseteq \varphi_3}{\varphi_1 \sqsubseteq \varphi_3} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2 \quad \varphi_3 \sqsubseteq \varphi_4}{\varphi_1;\varphi_3 \sqsubseteq \varphi_2;\varphi_4}$$

$$\frac{\varphi_1 \sqsubseteq \varphi_2 \quad \varphi_3 \sqsubseteq \varphi_4}{\varphi_1 + \varphi_3 \sqsubseteq \varphi_2 + \varphi_4} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2}{\text{spawn } \varphi_1 \sqsubseteq \text{spawn } \varphi_2} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2}{\text{rec}\beta.\varphi_1 \sqsubseteq \text{rec}\beta.\varphi_2}$$

$$\frac{\widehat{\tau} \leq \widehat{\tau}' \quad \widehat{\tau}' \leq \widehat{\tau} \quad r \subseteq r'}{\widehat{\tau} \text{ chan } r \sqsubseteq \widehat{\tau}' \text{ chan } r'} \qquad \frac{r_1 \subseteq r_2 \quad \widehat{\tau}_1 \leq \widehat{\tau}_2}{r_1!\widehat{\tau}_1 \sqsubseteq r_2!\widehat{\tau}_2} \qquad \frac{r_1 \subseteq r_2 \quad \widehat{\tau}_2 \leq \widehat{\tau}_1}{r_1?\widehat{\tau}_1 \sqsubseteq r_2?\widehat{\tau}_2}$$

$$\varphi_1;(\varphi_2;\varphi_3) \sqsubseteq (\varphi_1;\varphi_2);\varphi_3 \qquad (\varphi_1;\varphi_2);\varphi_3 \sqsubseteq \varphi_1;(\varphi_2;\varphi_3)$$

$$(\varphi_1 + \varphi_2);\varphi_3 \sqsubseteq (\varphi_1;\varphi_3) + (\varphi_1;\varphi_3) \qquad (\varphi_1;\varphi_3) + (\varphi_1;\varphi_3) \sqsubseteq (\varphi_1 + \varphi_2);\varphi_3$$

$$\varphi \sqsubseteq \Lambda;\varphi \qquad \Lambda;\varphi \sqsubseteq \varphi \qquad \varphi \sqsubseteq \varphi;\Lambda \qquad \varphi;\Lambda \sqsubseteq \varphi$$

$$\varphi_1 \sqsubseteq \varphi_1 + \varphi_2 \qquad \varphi_2 \sqsubseteq \varphi_1 + \varphi_2 \qquad \varphi + \varphi \sqsubseteq \varphi$$

$$\text{rec}\beta.\varphi \sqsubseteq \varphi[\beta \mapsto \text{rec}\beta.\varphi] \qquad \varphi[\beta \mapsto \text{rec}\beta.\varphi] \sqsubseteq \text{rec}\beta.\varphi$$

Example (1)

```
let node = fnF f => fnI inp => fnO out =>
             spawn ((funH h d => let v = receive inp
                                   in send (f v) on out; h d) ())
in ···
```

Type for node:

$$\forall\, 'a, 'b, '1, ''1, ''2.\ \underbrace{('a \xrightarrow{'1} 'b)}_{\texttt{f}} \xrightarrow{\Lambda} \underbrace{('a\ \texttt{chan}\ ''1)}_{\texttt{inp}} \xrightarrow{\Lambda} \underbrace{('b\ \texttt{chan}\ ''2)}_{\texttt{out}} \xrightarrow{\varphi} \texttt{unit}$$

where $\varphi = \mathsf{spawn}(\mathsf{rec}\ '2.\ (''1?'a;\ '1;\ ''2!'b;\ '2))$

Example (2)

```
let node = ···
in fun_P pipe fs => fn_I inp => fn_O out =>
        if isnil fs then node (fn_X x => x) inp out
        else let ch = channel_C
             in (node (hd fs) inp ch; pipe (tl fs) ch out)
```

Type for `pipe`:

$$\forall\ 'a, '1, ''1, ''2.$$

$$\underbrace{(('a \xrightarrow{'1} 'a)\ \texttt{list})}_{\texttt{fs}} \xrightarrow{\Lambda} \underbrace{('a\ \texttt{chan}\ (''1 \cup \{C\}))}_{\texttt{inp, ch}} \xrightarrow{\Lambda} \underbrace{('a\ \texttt{chan}\ ''2)}_{\texttt{out}} \xrightarrow{\varphi} \texttt{unit}$$

where $\varphi =$ $\text{rec}\ '2.\ (\underbrace{\text{spawn}(\text{rec}\ '3.((''1 \cup \{C\})?'a;\ \Lambda;\ ''2!'a;\ '3))}_{\texttt{node (fn x => x) ...}}$

$$+\ 'a\ \text{chan}\ C;\ \underbrace{\text{spawn}(\text{rec}\ '4.\ ((''1 \cup \{C\})?'a;\ '1;\ C!'a; '4));}_{\texttt{node (hd fs) ...}}\ '2)$$