Practice 1b

(Function $q \to q$ (Function $r \to r$ (Function $s \to s$ 4)))

(Function $q \to q$ (Function $r \to r$ (Function $s \to s$ 4))) v_1

v_1 (Function $r \to r$ (Function $s \to s$ 4))

(Function $f \to e_2$) (Function $r \to r$ (Function $s \to s$ 4))

(Function $f \to f\ e_3$) (Function $r \to r$ (Function $s \to s$ 4))

(Function $r \to r$ (Function $s \to s$ 4)) e_3

(Function $g \to e_4$) (Function $s \to s$ 4)

(Function $g \to g$ (Function $h \to e_5$)) (Function $s \to s$ 4)

(Function $s \to s$ 4) (Function $h \to e_5$)

(Function $h \to e_5$) 4

Function $f \to f$ (Function $g \to$ (g (Function $h \to h$)))

v_1 = Function $f \to e_2$

$e_2 = f\ e_3$

e_3 = Function $g \to e_4$

$e_4 = g$ (Function $h \to e_5$)

$e_5 = h$

$e_4 = g\ e_4'$

(Function $s \to s$ 4) e_4'

Language Design Question

FbT — Fb w/ n-tuples

- Syntax for construction and destruction
- Operational semantics
- TFbT rules
- EFbT rules
  - inference
  - closure
  - consistency

$$e ::= \cdots \mid (e, \ldots, e) \mid \text{Get } n\ e$$

$$v ::= \cdots \mid (v, \ldots, v) \qquad \tau ::= \cdots \mid (\tau_1, \ldots, \tau_n)$$

$$n ::= 0 \mid 1 \mid \cdots$$

Get True (0, 1)

!!

$e_1, e_2, \ldots, e_n \Rightarrow v_1, v_2, \ldots, v_n$

$e_1 \Rightarrow v_1 \quad \cdots \quad e_n \Rightarrow v_n$

$$\frac{\forall i \in \{1, \ldots, n\}.\ e_i \Rightarrow v_i}{(e_1, e_2, \ldots, e_n) \Rightarrow (v_1, v_2, \ldots, v_n)}$$

$$\frac{e \Rightarrow (v_1, v_2, \ldots, v_m) \qquad 1 \leq n \leq m}{\text{Get } n\ e \Rightarrow v_n}$$

$$\frac{\forall i \in \{1, \ldots, n\}.\ \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_1, \ldots, e_n) : (\tau_1, \ldots, \tau_n)}$$

$\emptyset \vdash \text{Get } 0\ (4, 3) : \text{Int}$

$\emptyset \vdash \text{Get } 2\ (4, 3) :$!!

$$\frac{\Gamma \vdash e : (\tau_1, \ldots, \tau_m) \qquad 1 \leq n \leq m}{\Gamma \vdash \text{Get } n\ e : \tau_n}$$

Let f = Function w → (w, w, w, 0) In

Get 2 (f 3)

$$\frac{\Gamma \vdash e_1 : \tau_1 \setminus E_1 \quad \cdots \quad \Gamma \vdash e_n : \tau_n \setminus E_n}{\Gamma \vdash (e_1, \ldots, e_n) : (\tau_1, \ldots, \tau_n) \setminus \bigcup_{1 \leq i \leq n} E_i}$$

$$\frac{\emptyset \vdash 3 : \text{Int} \setminus \emptyset \qquad \emptyset \vdash 4 : \text{Int} \setminus \emptyset}{\emptyset \vdash (3, 4) :}$$

$$\tau ::= \text{Int} \mid \text{Bool} \mid \tau \to \tau \mid (\tau, \ldots, \tau) \mid \alpha$$

$$\frac{\Gamma \vdash e : \tau \setminus E \qquad \alpha \text{ fresh} \qquad m \geq n \geq 1}{\Gamma \vdash \text{Get } n\ e : \alpha \setminus E \cup \{\tau = (\tau_1, \ldots, \tau_m), \alpha = \tau_n\}}$$

When $(\tau_1, \ldots, \tau_n) = (\tau_1', \ldots, \tau_n')$ then $\tau_1 = \tau_1', \ldots, \tau_n = \tau_n'$

$\{x : 'a\} \vdash \text{Get } n\ x : 'b \setminus \{'a[n] = 'b\}$

NOT GOING TO BE A ?

EFb & type variables

Bad Plus $$\frac{\Gamma \vdash e_1 : Int \setminus E_1 \quad \Gamma \vdash e_2 : Int \setminus E_2}{\Gamma \vdash e_1 + e_2 : Int \setminus E_1 \cup E_2}$$

$$\underbrace{f\ x}_{\alpha} + \underbrace{1}_{Int}$$

$\alpha = Int$

$$\frac{\Gamma \vdash e_1 : \tau_1 \setminus E_1 \quad \Gamma \vdash e_2 : \tau_2 \setminus E_2}{\Gamma \vdash e_1\ e_2 : \alpha \setminus \{\tau_1 = \tau_2 \to \alpha\}}$$

$Int \setminus \emptyset$

$'a \setminus \{'a = Int\}$

Plus $$\frac{\Gamma \vdash e_1 : \tau_1 \setminus E_1 \quad \Gamma \vdash e_2 : \tau_2 \setminus E_2}{\Gamma \vdash e_1 + e_2 : Int \setminus E_1 \cup E_2 \cup \{\tau_1 = Int, \tau_2 = Int\}}$$

Practice Problem 2

$v ::= \ldots \mid \text{Some } v \mid \text{None}$

$\tau ::= \ldots \mid \text{Maybe } \tau$

TFbM

$\boxed{(\text{Function } x \to \text{With } x \text{ As Some } y \to y \mid \text{None} \to 0)}$ (Some 8)

$\boxed{\text{Maybe Int} \to \text{Int}}$

STFbM

$\tau ::= \ldots \mid \text{Some } \tau \mid \text{None} \mid \text{Maybe } \tau$

$$\frac{\tau <: \tau'}{\text{Some } \tau <: \text{Maybe } \tau'}$$

[5] : int list

[] : string list

$$\frac{}{\Gamma \vdash \text{None} : \text{Maybe } \tau}$$

proofs

$$\frac{}{\emptyset \vdash \text{None} : \text{Maybe Bool}}$$

$$\frac{}{\emptyset \vdash \text{None} : \text{Maybe Int}}$$

rule

$$\frac{\alpha \text{ is fresh}}{\Gamma \vdash \text{None} : \text{Maybe } \alpha \setminus \emptyset}$$

Operational Equivalence

If 4 = (Let x = Ref 0 In
Let f = Function y → !y In
Let g = Function z → z := 4 In

Let n = f x In
Let m = g x In
n

Let n = f x In
Let m = g x In
f x

← g x changes state?

) Then 0 0 Else 0

Let f = Function x → # Raise Foo 0 In

$\boxed{0 \quad 0}$

$\boxed{\text{Let } z = f \ 0 \text{ In } 0 \ 0}$

Subtyping

$$\frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\underbrace{\tau_1 \to \tau_2}_{\text{bar}} <: \underbrace{\tau_1' \to \tau_2'}_{\text{foo}}}$$

$\tau ::= \ldots \mid \{\ell : \tau, \ldots, \ell : \tau\}$

$\tau_1 <: \tau_2$ iff

τ_1 guarantees more than τ_2

$\{\}$ TFbR: a record with exactly 0 fields

STFbR: a record with at least 0 fields

$\{x : \text{Int}\}$ TFbR: a record with exactly 1 field $(x : \text{Int})$

STFbR: a record with at least 1 field $(x : \text{Int})$

$\{x : \text{Int}\} <: \{\}$

know more

Animal

:>

Dog

:>

Poodle

$$\frac{\text{Poodle} <: \text{Dog} \qquad \text{Dog} <: \text{Animal}}{\text{Animal} \to \text{Poodle} <: \text{Dog} \to \text{Dog}}$$

$\{x : \text{Int}\} \not<: \{y : \text{Int}\}$

$\{x : \text{Int}, y : \text{Int}\} <: \{x : \text{Int}\}$

$\{\} <: \{\}$

$\{x : \text{Int}\} \to \{x : \text{Int}\} <: \{x : \text{Int}\} \to \{\}$

$\{\} \to \{x : \text{Int}\} <: \{x : \text{Int}\} \to \{x : \text{Int}\}$

$$\overline{\tau <: \top}$$ ← $\top$ is all values

$$\overline{\bot <: \tau}$$

$\emptyset$

(for fun)

$\top \to \bot <: \tau_1 \to \tau_2 <: \bot \to \top$

input → output

Evaluation

(Function a → Function b → Function c → c a b 1) (Function d → d + 1) True •

(Function x → Function y → Function z →
x (x (x (x z))))

(Function b → Function c → c (Function d → d+1) b 1) True •

$\bullet \; \underbrace{(\text{Function } d \to d+1)}_{x} \; \underbrace{\text{True}}_{y} \; \underbrace{1}_{z}$

Proofs

- Induction on height of proof tree <u>generally</u> more powerful (when you can)
- If case analyzing on form of an expr, <u>usually</u> want induction on height of expr
- 〃 proof rule, 〃 proof

$\forall e. \exists v.\ e \Rightarrow v$ Ind on $h(e)$, case analysis on e

If $e \Rightarrow v$ and $e : \tau$ then $v : \tau$ Ind on $h(e \Rightarrow v)$, case on rule

In class for soundness: ind on $h(e \Rightarrow v)$
not $h(\Gamma \vdash e : \tau)$

Substitution Lemma

$$\underbrace{\Gamma, x : \tau \vdash e : \tau'}_{h(\cdot) = n} \quad \text{and} \quad \Gamma \vdash v : \tau \quad \text{then} \quad \underbrace{\Gamma \vdash e[v/x] : \tau'}_{h(\cdot) = ?}$$

$$h\left(\frac{}{\Gamma, x : Int \to Int \vdash x : Int \to Int}\right) = 0$$

$v = $ Function $x : Int \to x$

$$h\left(\frac{\Gamma, x : Int \vdash x : Int}{\Gamma \vdash \text{Function } x : Int \to x}\right) = 1$$

Exceptions

FbX

$k ::=$ (identifiers)

NEVER in a variable

$v ::= \cdots \mid \#k\ v \mid \text{Raise}\ v$

$e ::= \cdots \mid \#k\ e \mid \text{Raise}\ e \mid \text{Try}\ e\ \text{With}\ \#k\ x \rightarrow e$

important but boring

$$\frac{e \Rightarrow v}{\#k\ e \Rightarrow \#k\ v}$$

$$\frac{e \Rightarrow v}{\text{Raise}\ e \Rightarrow \text{Raise}\ v}$$

$$\frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2 \quad v_1, v_2 \in \mathbb{Z} \quad v = \text{sum of } v_1, v_2}{e_1 + e_2 \Rightarrow v}$$

lots of these

$$\frac{e_1 \Rightarrow \text{Raise}\ v_1}{e_1 + e_2 \Rightarrow \text{Raise}\ v_1}$$

$$\frac{e_1 \Rightarrow v_1 \quad v_1 \text{ not of form Raise } v_1' \quad e_2 \Rightarrow \text{Raise}\ v_2}{e_1 + e_2 \Rightarrow \text{Raise}\ v_2}$$

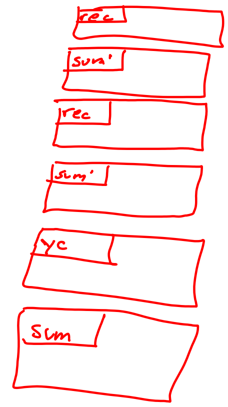
$$\frac{e_1 \Rightarrow \text{True} \quad e_2 \Rightarrow \text{Raise}\ v}{\text{If}\ e_1\ \text{Then}\ e_2\ \text{Else}\ e_3 \Rightarrow \text{Raise}\ v}$$

FbX ÷

$$\frac{e_1 \Rightarrow v_1 \quad v_1 \text{ not of form Raise } v_1' \quad e_2 \Rightarrow 0}{e_1 / e_2 \Rightarrow \text{Raise}\ \#\text{DivZero}\ 0}$$

Y-combinator

```
Let sum' =
  (Function self -> Function n -> If n=0 Then 0 Else n + self self (n-1))
In
Let sum = sum' sum' In
sum 5
```



function (like sum')

value can be given to f

```
Let yc = Function f -> Function x ->
  Let recurser = Function self -> Function x' ->
    f (self self) x'
  In
  f (recurser recurser) x
In
```

recursion function (same behavior as sum') takes 1 arg

number

```
Let sum' =
  Function recurse -> Function n -> If n=0 Then 0 Else n + recurse (n-1)
In
Let sum = yc sum' In
sum 5
```

arg of foo

```
Let y2 = Function f1 -> Function f2 -> Function x ->
  Let recurser1 = Function self -> Function other -> Function x' ->
    f1 (self self other) (other other self) x'
  In
  Let recurser2 = Function self -> Function other -> Function x' ->
    f2 (other other self) (self self other) x'
  In

  f1 (recurser1 recurser1 recurser2) (recurser2 recurser2 recurser1) x
In
Let foo = y2 foo' bar' In
foo 5
```

fn taking same type as x, returning what f1 returns