Answer Refinement Modification

Refinement Type System for Algebraic Effects and Handlers

POPL 2024

Fuga Kawamata
Waseda University

Hiroshi Unno
University of Tsukuba

Taro Sekiyama
National Institute of Informatics

Tachio Terauchi
Waseda University

Algebraic Effects and Handlers

- A mechanism to structure programs with effects in a modular way
- Can represent many kinds of effects
  - exception, state, I/O, nondeterminism, concurrency, etc.

```
handle
  Set 1; let a1 = Get () in
  Set 2; let a2 = Get () in
  a1 + a2
with
  | return x   -> λs. x
  | Set  v, k -> λs. k () v
  | Get (), k -> λs. k  s s
```

```
handle
  if Decide () then 1 else 2
with
  | return x      -> x
  | Decide (), k -> k true + k false
```

operation

handler

- Have been adopted in OCaml 5

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()


with h handle (Tick (); Tock ())
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())    ⟶  1 :: with h handle ((); Tock ())
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()


with h handle (Tick (); Tock ())      ⟶  1 :: with h handle ((); Tock ())

                                      ⟶  1 :: with h handle (Tock ())
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())    ⟶  1 :: with h handle ((); Tock ())

                                    ⟶  1 :: with h handle (Tock ())
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())    ⟶  1 :: with h handle ((); Tock ())
                                     ⟶  1 :: with h handle (Tock ())
                                     ⟶  1 :: 0 :: with h handle ()
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())    ⟶  1 :: with h handle ((); Tock ())
                                    ⟶  1 :: with h handle (Tock ())
                                    ⟶  1 :: 0 :: with h handle ()
```

Algebraic Effects and Handlers

- **Example**

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())    ⟶  1 :: with h handle ((); Tock ())
                                    ⟶  1 :: with h handle (Tock ())
                                    ⟶  1 :: 0 :: with h handle ()
                                    ⟶  1 :: 0 :: []
                                    =  [1; 0]
```

Algebraic Effects and Handlers

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

int list

int list

int list

unit → int list

⊢ `with h handle (Tick();Tock())` : int list

Algebraic Effects and Handlers

value types $T ::= B \mid T \to C$
computation types $C ::= \Sigma \rhd T$
signatures $\Sigma ::= \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi}\}_i$

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi}\}_i \rhd T_0 \qquad \Gamma, x : T_0 \vdash e_r : \boldsymbol{C} \qquad (\Gamma, x : T_{ai}, k : T_{bi} \to \boldsymbol{C} \vdash e_i : \boldsymbol{C})_i}{\Gamma \vdash \textbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \textbf{handle}\ e : \boldsymbol{C}}$$

$$\frac{\Sigma \ni \mathrm{Op} : T_a \twoheadrightarrow T_b \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd T_b}$$

$\Gamma \vdash e : C$

Refinement Types

$$\{x : B \mid \phi\}$$

Formula

Base Type (int, bool etc.)

- Types endowed with predicates
- E.g. $1 + 2 : \{x : \mathrm{int} \mid x = 3\}$
  $1 + 2 : \{x : \mathrm{int} \mid x > 0\}$
- Can represent more specific properties
  - A tool of program verification

Contribution

Refinement type system for algebraic effect handlers

$\vdash$ `with h handle (Tick (); Tock ())` $: \{z : \mathrm{int\ list} \mid z = [1; 0]\}$

- Enables precise verification of algebraic effect handlers
- Can be built on top of existing languages
  - Our implementation for OCaml 5 programs

Challenge

```
let h = handler
| return x -> x
| Tick (), k -> 1 :: k ()
| Tock (), k -> 0 :: k ()
```

The precise types depend on the **order** of the operations

```
with h handle
  (Tick (); Tock ())
(* ==> [1; 0] *)
```

$: \{z\colon \mathrm{int\ list} \mid z = [1;0]\}$

```
with h handle
  (Tock (); Tick ())
(* ==> [0; 1] *)
```

$: \{z\colon \mathrm{int\ list} \mid z = [0;1]\}$

Challenge

```
let h = handler
| return x -> x
| Decide (), k -> k true - k false


with h handle
  (if Decide () then n1 else n2)
(* ==> n1 - n2 *)
```

$: \{z\colon \mathrm{int} \mid z = n_1 - n_2\}$

The precise type depends on
which branch returns **what value**

Challenge

$$\frac{??}{\Gamma \vdash \textbf{with } h \textbf{ handle } e : ??}$$

Challenge

Need to be aware of the **structure** of e
to obtain precise types

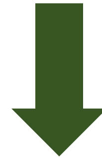
$$\frac{??}{\Gamma \vdash \textbf{with}\ h\ \textbf{handle}\ e : ??}$$

Challenge

Need to be aware of the **structure** of e
to obtain precise types

$$??$$

$$\Gamma \vdash \textbf{with}\ h\ \textbf{handle}\ e : ??$$



We adopt

Answer Type Modification

[Danvy+90; Materzok+11]

Answer Types

- Types of **contexts**
  - = return types of **continuations**
  - = types of **the closest outer handling constructs (delimiters)**

$$\mathcal{E}[\,\textbf{with}\ h\ \textbf{handle}\ \mathcal{F}[e]\,]$$

evaluation context

handler-free evaluation context

Answer type of e = type of $\textbf{with}\ h\ \textbf{handle}\ \mathcal{F}[\,]$

$$\mathcal{F} ::= [\,] \mid \textbf{let}\ x = \mathcal{F}\ \textbf{in}\ e$$
$$\mathcal{E} ::= [\,] \mid \textbf{let}\ x = \mathcal{E}\ \textbf{in}\ e \mid \textbf{with}\ h\ \textbf{handle}\ \mathcal{E}$$

Answer Types in Ordinary Type Systems

value types $T ::= B \mid T \to C$
computation types $C ::= \Sigma \rhd T$
signatures $\Sigma ::= \{\mathrm{Op}_i \colon T_{ai} \twoheadrightarrow T_{bi}\}_i$

answer type

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i \colon T_{ai} \twoheadrightarrow T_{bi}\}_i \rhd T_0 \qquad \Gamma, x \colon T_0 \vdash e_r : \boldsymbol{C} \qquad (\Gamma, x \colon T_{ai}, k \colon T_{bi} \to \boldsymbol{C} \vdash e_i : \boldsymbol{C})_i}{\Gamma \vdash \mathbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \mathbf{handle}\ e : \boldsymbol{C}}$$

$$\frac{\Sigma \ni \mathrm{Op} \colon T_a \twoheadrightarrow T_b \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd T_b}$$

$\boxed{\Gamma \vdash e : C}$

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (Tick (); Tock ())
```

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

unit → {z: **int list** | z = [ ]}

`with h handle (   )` returns [ ]

```
with h handle (Tick (); Tock ())
→ 1 :: with h handle ((); Tock ())
→ 1 :: with h handle (Tock ())
→ 1 :: 0 :: with h handle ()
→ 1 :: 0 :: []
```

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

unit → $\{z\colon \textbf{int list} \mid z = [\,]\}$

$\{z\colon \textbf{int list} \mid z = [0]\}$

```
with h handle (      ) returns []
1 ::                   takes [0]
```

```
with h handle (Tick (); Tock ())
→ 1 :: with h handle ((); Tock ())
→ 1 :: with h handle (Tock ())
→ 1 :: 0 :: with h handle ()
→ 1 :: 0 :: []
```

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

Initial answer type

unit → $\{z\colon \text{int list} \mid z = [\,]\}$

Final answer type

$\{z\colon \text{int list} \mid z = [0]\}$

Answer type modification (ATM)

Answer refinement modification (ARM)

```
with h handle (Tick (); Tock ())
→ 1 :: with h handle ((); Tock ())
→ 1 :: with h handle (Tock ())
→ 1 :: 0 :: with h handle ()
→ 1 :: 0 :: []
```

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

unit $\to \{z : \mathbf{int\ list} \mid z = [0]\}$

```
with h handle (Tick (); Tock ())
→ 1 :: with h handle ((); Tock ())
→ 1 :: with h handle (Tock ())
→ 1 :: 0 :: with h handle ()
→ 1 :: 0 :: []
```

Answer Type Modification

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

unit $\to$ $\{z\colon \mathbf{int\ list} \mid z = [0]\}$

$\{z\colon \mathbf{int\ list} \mid z = [1; 0]\}$

```
with h handle (Tick (); Tock ())
→ 1 :: with h handle ((); Tock ())
→ 1 :: with h handle (Tock ())
→ 1 :: 0 :: with h handle ()
→ 1 :: 0 :: []
```

Type Syntax

value types	$T ::= \{x : B \mid \phi\} \mid T \to C$
computation types	$C ::= \Sigma \rhd T \,/\, \boldsymbol{S}$
signatures	$\Sigma ::= \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \,/\, \boldsymbol{C_{1i}} \Rightarrow \boldsymbol{C_{2i}}\}_i$
control effects	$\boldsymbol{S} ::= \square \mid \boldsymbol{C_1} \Rightarrow \boldsymbol{C_2}$

Typing Rules

- **Example**

$$\Sigma = \{\ \text{Tick: unit} \twoheadrightarrow \text{unit} \,/\, \{z\text{: int list} \mid z = [0]\} \Rightarrow \{z\text{: int list} \mid z = [1;0]\},$$
$$\text{Tock: unit} \twoheadrightarrow \text{unit} \,/\, \{z\text{: int list} \mid z = [\,]\} \Rightarrow \{z\text{: int list} \mid z = [0]\}\ \}$$

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (
```

`Tick ();` $: \Sigma \rhd \text{unit} \,/\, \{z\text{: int list} \mid z = [0]\} \Rightarrow \{z\text{: int list} \mid z = [1;0]\}$

`Tock ()` $: \Sigma \rhd \text{unit} \,/\, \{z\text{: int list} \mid z = [\,]\} \Rightarrow \{z\text{: int list} \mid z = [0]\}$

```
)
```

Typing Rules

- **Example**

$$\Sigma = \{\, \mathrm{Tick} \colon \mathrm{unit} \twoheadrightarrow \mathrm{unit} \,/\, \{z \colon \mathrm{int\ list} \mid z = [0]\} \Rightarrow \{z \colon \mathrm{int\ list} \mid z = [1; 0]\},$$
$$\mathrm{Tock} \colon \mathrm{unit} \twoheadrightarrow \mathrm{unit} \,/\, \{z \colon \mathrm{int\ list} \mid z = [\,]\} \Rightarrow \{z \colon \mathrm{int\ list} \mid z = [0]\} \,\}$$

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()

with h handle (
  Tick ();
  Tock ()
)
```

$$: \Sigma \rhd \mathrm{unit} \,/\, \{z \colon \mathrm{int\ list} \mid z = [\,]\} \Rightarrow \{z \colon \mathrm{int\ list} \mid z = [1; 0]\}$$

Typing Rules

- **Example**

$$\Sigma = \left\{ \begin{array}{l} \text{Tick}: \text{unit} \twoheadrightarrow \text{unit} \,/\, \{z: \text{int list} \mid z = [0]\} \Rightarrow \{z: \text{int list} \mid z = [1;0]\}, \\ \text{Tock}: \text{unit} \twoheadrightarrow \text{unit} \,/\, \{z: \text{int list} \mid z = [\,]\} \Rightarrow \{z: \text{int list} \mid z = [0]\} \end{array} \right\}$$

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()
```

```
with h handle (

  Tick ();

  Tock ()

)
```

$: \{z: \text{int list} \mid z = [1;0]\}$

Typing Rules

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \;/\; C_{1i} \Rightarrow C_{2i}\}_i \rhd T_0 \;/\; C_1 \Rightarrow C_2 \qquad \Gamma, x : T_0 \vdash e_r : C_1 \qquad (\Gamma, x : T_{ai}, k : T_{bi} \rightarrow C_{1i} \vdash e_i : C_{2i})_i}{\Gamma \vdash \textbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \textbf{handle}\ e : C_2}$$

$$\frac{\Sigma \ni \mathrm{Op} : T_a \twoheadrightarrow T_b \;/\; C_1 \Rightarrow C_2 \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd T_b \;/\; C_1 \Rightarrow C_2}$$

$\boxed{\Gamma \vdash e : C}$

Typing Rules

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \;/\; C_{1i} \Rightarrow C_{2i}\}_i \rhd T_0 \;/\; C_1 \Rightarrow C_2 \qquad \Gamma, x : T_0 \vdash e_r : C_1 \quad (\Gamma, x : T_{ai}, k : T_{bi} \to C_{1i} \vdash e_i : C_{2i})_i}{\Gamma \vdash \textbf{with}\; \{x \mapsto e_r, (\mathrm{Op}_i\; x, k \mapsto e_i)_i\}\; \textbf{handle}\; e : C_2}$$

$$\frac{\Sigma \ni \mathrm{Op} : T_a \twoheadrightarrow T_b \;/\; C_1 \Rightarrow C_2 \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\; v : \Sigma \rhd T_b \;/\; C_1 \Rightarrow C_2}$$

$\Gamma \vdash e : C$

Typing Rules

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \;/\; C_{1i} \Rightarrow C_{2i}\}_i \rhd T_0 \;/\; C_1 \Rightarrow C_2 \qquad \Gamma, x : T_0 \vdash e_r : C_1 \quad (\Gamma, x : T_{ai}, k : T_{bi} \to C_{1i} \vdash e_i : C_{2i})_i}{\Gamma \vdash \textbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \textbf{handle}\ e : C_2}$$

$$\frac{\Sigma \ni \mathrm{Op} : T_a \twoheadrightarrow T_b \;/\; C_1 \Rightarrow C_2 \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd T_b \;/\; C_1 \Rightarrow C_2}$$

$\boxed{\Gamma \vdash e : C}$

In the Paper

- A more sophisticated type system and the proof of the type safety
  - $\Sigma ::= \{\mathrm{Op}_i : \forall X : \bar{B}. x : T_{ai} \twoheadrightarrow T_{bi} \mathbin{/} y. C_{1i} \Rightarrow C_{2i}\}_i$
    - Predicate polymorphism
    - Argument-dependent continuation types
- Implementation
  - On top of OCaml 5.0
  - Experiments on 50+ examples

https://github.com/hiroshi-unno/coar

- Another way of verification via CPS transformation
  - Removal of handlers + verification with existing refinement type systems
  - Bidirectionally type-preserving CPS transformation
    - Needs type annotations on source programs

Appendix

Extension 1: Dependent Types for Continuations

value types $T ::= B \mid T \to C$
computation types $C ::= \Sigma \rhd T \,/\, S$
signatures $\Sigma ::= \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \,/\, y.\, C_{1i} \Rightarrow C_{2i}\}_i$
control effects $S ::= \square \mid y.\, C_1 \Rightarrow C_2$

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : T_{ai} \twoheadrightarrow T_{bi} \,/\, y.\, C_{1i} \Rightarrow C_{2i}\}_i \rhd T_0 \,/\, C_1 \Rightarrow C_2 \quad \Gamma, x : T_0 \vdash e_r : C_1 \quad (\Gamma, x : T_{ai}, k : (y : T_{bi}) \to C_{1i} \vdash e_i : C_{2i})_i}{\Gamma \vdash \mathbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \mathbf{handle}\ e : C_2}$$

$$\frac{\Sigma \ni \mathrm{Op} : T_a \twoheadrightarrow T_b \,/\, y.\, C_1 \Rightarrow C_2 \qquad \Gamma \vdash v : T_a}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd (T_b \,/\, y.\, C_1 \Rightarrow C_2)}$$

$\boxed{\Gamma \vdash e : C}$

Extension 1: Dependent Types for Continuations

- **Example**

```
let h = handler
| return x -> x
| Decide (), k -> k true - k false
```

$b\colon \mathrm{bool} \to \{z\colon \mathrm{int} \mid b \Rightarrow z = n_1 \land \neg b \Rightarrow z = n_2\}$

$b \Rightarrow z = n_1 \land \neg b \Rightarrow z = n_2\}$

$\Rightarrow \{z\colon \mathrm{int} \mid z = n_1 - n_2\}\}$

```
with h handle (
  let b = Decide () in
  if b then n1 else n2
)
```

$\{z\colon \mathrm{int} \mid z = n_1 - n_2\}$

$\{z\colon \mathrm{int} \mid b \Rightarrow z = n_1 \land \neg b \Rightarrow z = n_2\}$

Extension 2: Predicate Polymorphism

value types	$T ::= B \mid T \to C$
computation types	$C ::= \Sigma \rhd T \,/\, S$
signatures	$\Sigma ::= \{\mathrm{Op}_i : \forall X{:}\bar{B}.\, T_{ai} \twoheadrightarrow T_{bi} \,/\, C_{1i} \Rightarrow C_{2i}\}_i$
control effects	$S ::= \square \mid C_1 \Rightarrow C_2$

$$\frac{\Gamma \vdash e : \{\mathrm{Op}_i : \forall X{:}\bar{B}.\, T_{ai} \twoheadrightarrow T_{bi} \,/\, C_{1i} \Rightarrow C_{2i}\}_i \rhd T_0 \,/\, C_1 \Rightarrow C_2 \qquad \Gamma, x{:}T_0 \vdash e_r : C_1 \quad (\Gamma, X{:}\bar{B}, x{:}T_{ai}, k{:}T_{bi} \to C_{1i} \vdash e_i : C_{2i})_i}{\Gamma \vdash \textbf{with}\ \{x \mapsto e_r, (\mathrm{Op}_i\ x, k \mapsto e_i)_i\}\ \textbf{handle}\ e : C_2}$$

$$\frac{\Sigma \ni \mathrm{Op} : \forall X{:}\bar{B}.\, T_a \twoheadrightarrow T_b \,/\, C_1 \Rightarrow C_2 \qquad \Gamma \vdash v : T_a \qquad \Gamma \vdash A : \bar{B}}{\Gamma \vdash \mathrm{Op}\ v : \Sigma \rhd (T_b \,/\, C_1 \Rightarrow C_2)[A/X]}$$

$\boxed{\Gamma \vdash e : C}$

Extension 2: Predicate Polymorphism

- **Example**

$$\Sigma = \{\ \mathrm{Tick}\colon \forall X(\mathrm{int\ list}).\ \mathrm{unit} \twoheadrightarrow \mathrm{unit}\ /\ \{z\colon \mathrm{int\ list} \mid X(z)\} \Rightarrow \{z\colon \mathrm{int\ list} \mid \forall l. X(l) \Rightarrow z = 1 :: l\},\ \mathrm{Tock}\colon \forall Y(\mathrm{int\ list}).\ \mathrm{unit} \twoheadrightarrow \mathrm{unit}\ /\ \{z\colon \mathrm{int\ list} \mid Y(z)\} \Rightarrow \{z\colon \mathrm{int\ list} \mid \forall l. Y(l) \Rightarrow z = 0 :: l\}\ \}$$

```
let h = handler
  | return x -> []
  | Tick (), k -> 1 :: k ()
  | Tock (), k -> 0 :: k ()

with h handle (
  Tick ();
  Tock ();
  Tick ();
  Tock ()
)
```

$\{z\colon \mathrm{int\ list} \mid z = [1;0;1;0]\}$

$X \mapsto \lambda z. z = [0;1;0]$

$\{z\colon \mathrm{int\ list} \mid z = [0;1;0]\}$

$Y \mapsto \lambda z. z = [1;0]$

$\{z\colon \mathrm{int\ list} \mid z = [1;0]\}$

$X \mapsto \lambda z. z = [0]$

$\{z\colon \mathrm{int\ list} \mid z = [0]\}$

$Y \mapsto \lambda z. z = [\,]$

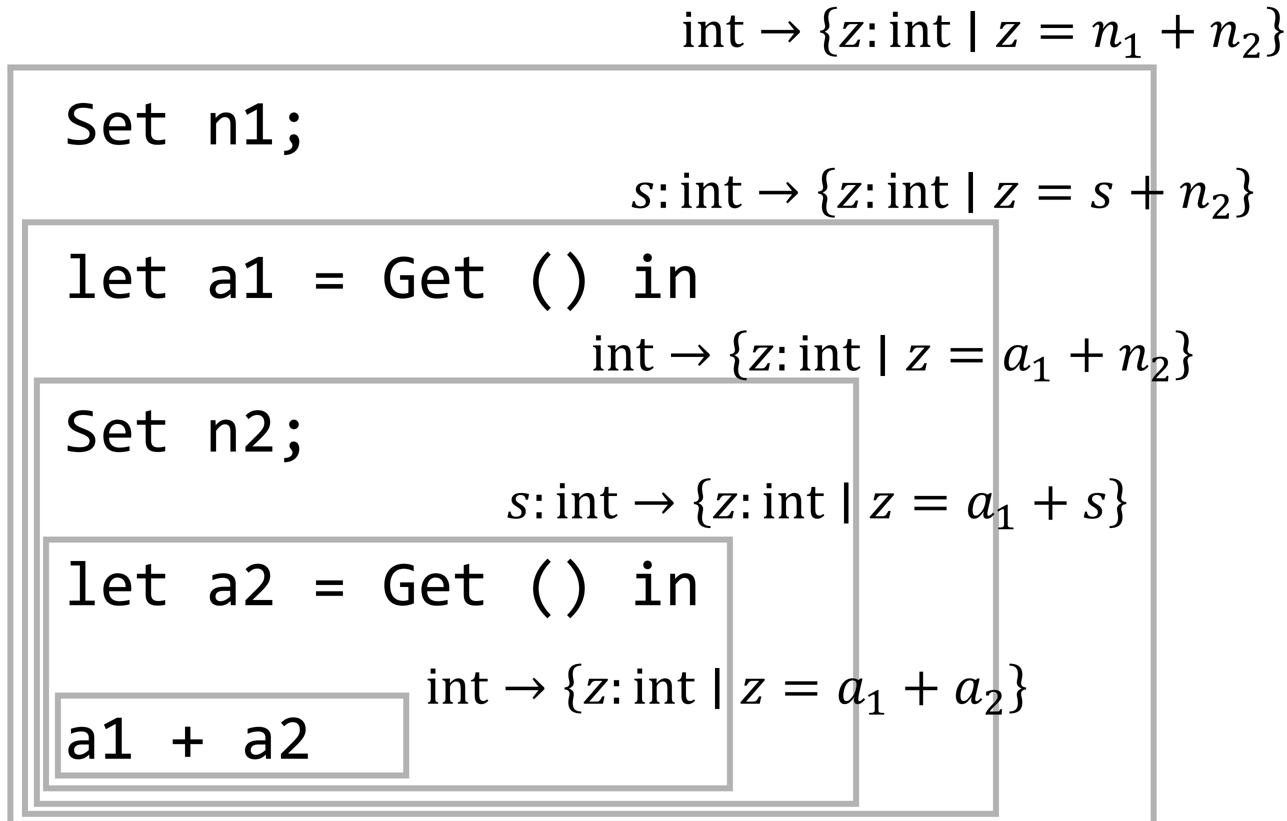
$\{z\colon \mathrm{int\ list} \mid z = [\,]\}$

Extension 2: Predicate Polymorphism

- **Example**

```
let h = handler
| return x  -> λs. x
| Set  v, k -> λs. k () v
| Get (), k -> λs. k  s s
```

```
with h handle
```



$$\Sigma = \Big\{\text{Set}{:}\ \forall X{:}\,(\text{int},\text{int}).\ x{:}\,\text{int} \twoheadrightarrow \text{unit}\ /\ s{:}\,\text{int} \to \{z{:}\,\text{int} \mid X(z,s)\} \Rightarrow \text{int} \to \{z{:}\,\text{int} \mid X(z,x)\},$$
$$\text{Get}{:}\ \forall Y{:}\,(\text{int},\text{int},\text{int}).\ \text{unit} \twoheadrightarrow \text{int}\ /\ y.\ s{:}\,\text{int} \to \{z{:}\,\text{int} \mid Y(z,s,y)\} \Rightarrow s'{:}\,\text{int} \to \{z{:}\,\text{int} \mid Y(z,s',s')\}\Big\}$$