Towards Compilation of Effect Handlers into System F

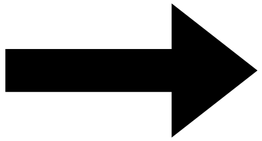
24M30090 川添裕功

Advantages of Algebraic Effects & Handlers

- Algebraic effects and handlers
  - Dealing with computational effects
    - State
    - Exception
    - ...
  - Separation of definition and implementation
    - Modularity

Challenges of Algebraic Effects & Handlers

- Algebraic effects and handlers
  - Implementation is not easy
    - dealing with delimited continuation



reduce difficulties to well-known type system

Algebraic Effects & Handlers

```
handle
  do b ← decide () in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

```
pickMax := handler {
  decide _ k ↦
    do xt ← k true in
    do xf ← k false in
    return max (xt, xf)}
# evaluate all possible
# results and return
# maximal result
```

Algebraic Effects & Handlers

- handler
  - determine
  - behavior of operation

```
pickMax := handler {
 decide _ k ↦
  do xt ← k true in
  do xf ← k false in
  return max (xt, xf)}
```

Algebraic Effects & Handlers

```
handle
  do b ← decide () in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

- handling effects

handle computation □
with handler __

Algebraic Effects & Handlers

```
handle
  do b ← decide () in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

operation call

```
pickMax := handler {
  decide _ k ↦
    do xt ← k true in
    do xf ← k false in
    return max (xt, xf)}
# evaluate all possible
# results and return
# maximal result
```

Algebraic Effects & Handlers

```
handle
  do b ← [          ] in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

captured continuation

```
pickMax := handler {
  decide _ k ↦
    do xt ← k true in
    do xf ← k false in
    return max (xt, xf)}
# evaluate all possible
# results and return
# maximal result
```

k := λ x. [x]

Algebraic Effects & Handlers

```
handle
   do b ← true in
   if b then return 1
   else return 2
   # depending on b
   # return 1 or 2
with pickMax
```

captured continuation

```
pickMax := handler {
   decide _ k ↦
      do xt ← k true in
      do xf ← k false in
      return max (xt, xf)}
# evaluate all possible
# results and return
# maximal result
```

k := λ x. x

Algebraic Effects & Handlers

```
handle
  do b ← false in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

captured continuation

```
pickMax := handler {
  decide _ k ↦
    do xt ← k true in
    do xf ← k false in
    return max (xt, xf)}
# evaluate all possible
# results and return
# maximal result
```

k := λ x. [x]

Algebraic Effects & Handlers

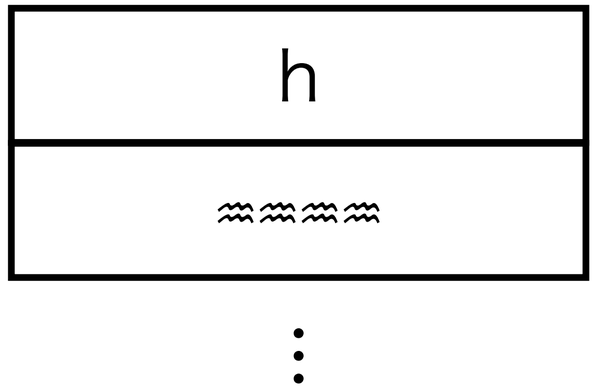
```
handle
  do b ← decide () in
  if b then return 1
  else return 2
  # depending on b
  # return 1 or 2
with pickMax
```

$\to^*$ return 2

Implementation is not easy

handling computation

handle .. with h



Implementation is not easy

operation call

.. op v ..

⋮

MOV .. .. ← instruction pointer

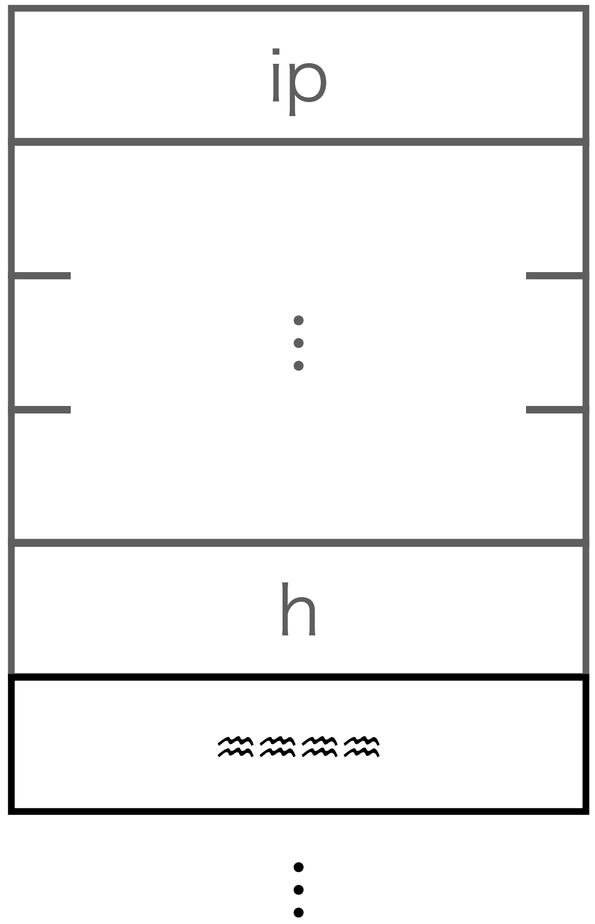
ADD .. .. (continuation)

CMP .. ..

JNE ..

⋮

cut



Implementation is not easy

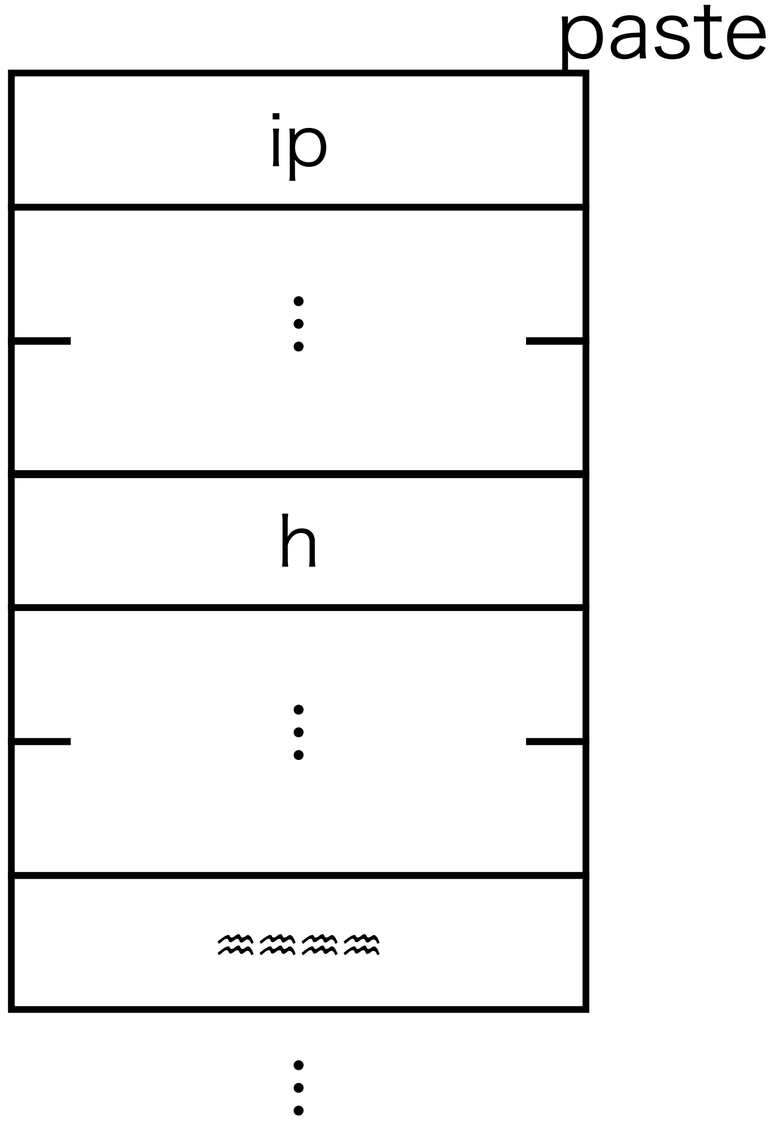
resuming

.. k v

⋮

POP ip ← instruction pointer

⋮



Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

have no effects

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v

may have effects

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v ← lift value to computation

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid \mathsf{do} \; x \leftarrow c_1 \; \mathsf{in} \; c_2$

$\mid \mathsf{if} \; v \; \mathsf{then} \; c_1 \; \mathsf{else} \; c_2$

$\mid v_1 \; v_2$

$\mid \mathsf{handle} \; c \; \mathsf{with} \; v$

operation call

handler $h := \mathsf{handler}\{\mathsf{return} \; x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2 ← sequencing

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v

condition

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2 ← clarify the evaluation order

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$ ← application

$\mid$ handle c with v

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v ← handling

handler $h := \mathsf{handler}\{\mathsf{return}\ x \mapsto c_r, \mathsf{op}_1\ x\ k \mapsto c_1, \ldots \mathsf{op}_n\ x\ k \mapsto c_n\}$

Fine-grain Call by Value

Value $v := x \mid \mathsf{false} \mid \mathsf{true} \mid \lambda x : A \,.\, c \mid h$

Computation $c :=$ return v

$\mid \mathsf{op}_i \; v \; y \,.\, c$

$\mid$ do $x \leftarrow c_1$ in c_2

$\mid$ if v then c_1 else c_2

$\mid v_1 \; v_2$

$\mid$ handle c with v

handler $h :=$ handler$\{$return $x \mapsto c_r, \mathsf{op}_1 \; x \; k \mapsto c_1, \ldots \mathsf{op}_n \; x \; k \mapsto c_n\}$

Fine-grain Call by Value

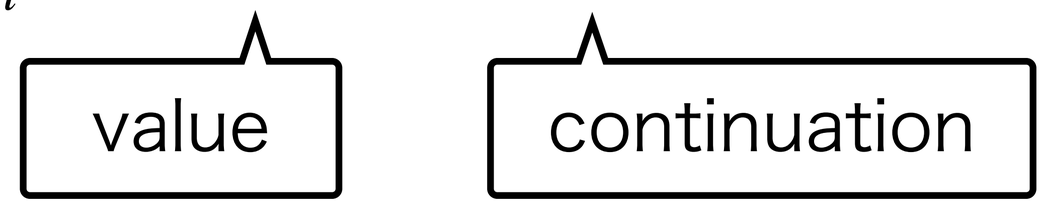
Value Type $A, B := \mathsf{bool} \mid A \to \underline{C} \mid \underline{C} \Rightarrow \underline{D}$

Computation Type $\underline{C}, \underline{D} := A\,!\{\mathsf{op}_1 : A_1 \to B_1, \ldots, \mathsf{op}_n : A_n \to B_n\}$

Generic Effects

operation call

op_i v $y\,.\,c$



$\underline{\mathrm{op_i}} = \lambda$ v. $\mathrm{op_i}$ v y.return y

Operational Semantics

$$\texttt{do } x \leftarrow (\texttt{op}_i\ v\ y.c)\ \texttt{in } c' \rightsquigarrow \texttt{op}_i\ v\ (y.\texttt{do } x \leftarrow c \texttt{ in } c')$$

$$\texttt{handle op}_i\ v\ y.c\ \texttt{with h} \rightsquigarrow [x \mapsto v,\ k \mapsto \lambda y.\texttt{handle } c \texttt{ with h}]c_i$$

where

$$h := \{\ldots\ \texttt{op}_i\ x\ k \mapsto c_i\ \ldots\}$$

Operational Semantics

```
handle
  do b ← decide () y.return y in
  if b then return 1
  else return 2
with pickMax
```

```
handle
  decide () (y. do b ← y in
  if b then return 1
  else return 2)
with pickMax
```

Implementation

Compiler

Translation to System F

Interpreter

Normalization

Translation to System F

$[\![\cdot]\!] : \text{Value Type} \to \text{Type}_{\mathsf{F}}$

$[\![\cdot]\!] : \text{Computation Type} \to \text{Type}_{\mathsf{F}}$

$[\![\cdot]\!] \Rightarrow \cdot : \text{Computation Type} \to \text{Type}_{\mathsf{F}} \to \text{Type}_{\mathsf{F}}$

Type Translation

$[\![\cdot]\!] : \Gamma \vdash_v A \to [\![A]\!]$

$\cdot : \cdot : \Gamma \vdash_c A\,!E \to [\![\Gamma]\!] \leq \Delta \to \Delta \vdash [\![A\,!E]\!] \Rightarrow B \to \Delta \vdash B$

Term Translation

Translation to System F

$[\![\cdot]\!] : \text{Value Type} \to \text{Type}_\mathsf{F}$

Type Translation

$[\![\cdot]\!] : \text{Computation Type} \to \text{Type}_\mathsf{F}$

$[\![\cdot]\!] \Rightarrow \cdot : \text{Computation Type} \to \text{Type}_\mathsf{F} \to \text{Type}_\mathsf{F}$

$[\![A \,!\, E]\!] = \quad \forall B \qquad [\![A \,!\, E \Rightarrow]\!] B \qquad \to B$

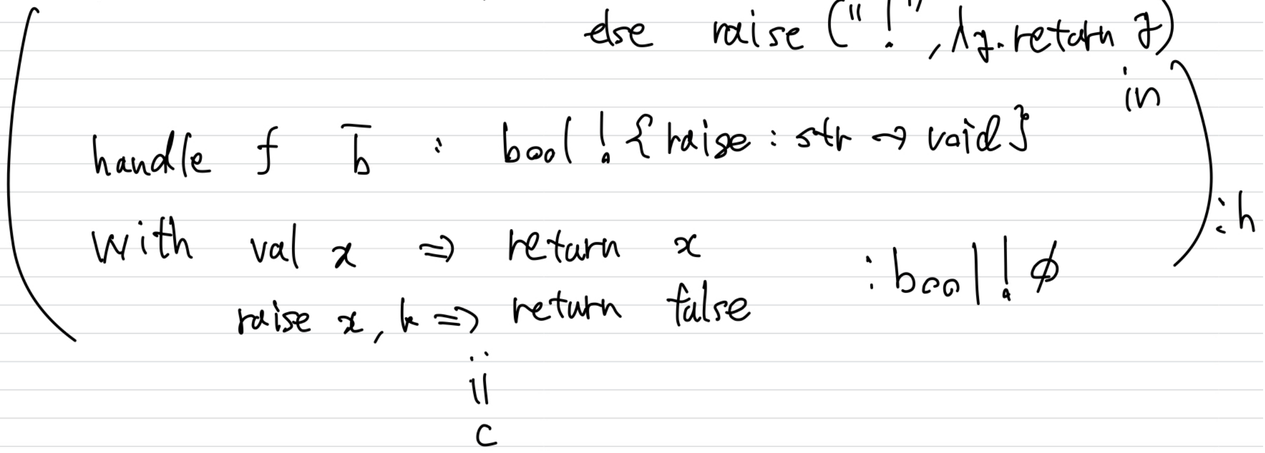
for all type B

takes handler

which returns type B

returns type B

Translation to System F



$c = d_a \; x \leftarrow c_1 \text{ in } c_2$

$h' = \langle \text{val} := \lambda f.\, c_2 : h$
$\quad \text{raise} := \lambda x.\lambda k.\, (\text{raise } x\ \lambda y.k\ y) : h$

$c : h = c_1 : h'$

$c_2 := \text{handle } f\ \bar{b} : \text{bool} !_a \{\text{raise} : \text{str} \to \text{void}\}$
$\quad \text{with val } x \Rightarrow \text{return } x$
$\quad \text{raise } x, k \Rightarrow \text{return false} \quad : \text{bool} !_a \phi$
$\quad \Vert$
$\quad d$

$\lceil d \rceil = \langle \text{val} := \lambda x.\lambda h.\, h.\text{val}\ x$
$\quad \text{raise} := \lambda x.\lambda k.\lambda h.\, h.\text{val false} \rangle$

$f\ \bar{b} : \lceil d \rceil$
$= \lceil f \rceil \lceil \bar{b} \rceil \lceil d \rceil$

$c_2 : h = f\ \bar{b}\ \lceil d \rceil\ h$

example of translation

Translation to System F

lemma: coherence

if $\Gamma \vdash c \rightsquigarrow c' : A\,!E, [\![\Gamma]\!] \vdash h : [\![A\,!E]\!] \Rightarrow B$ then

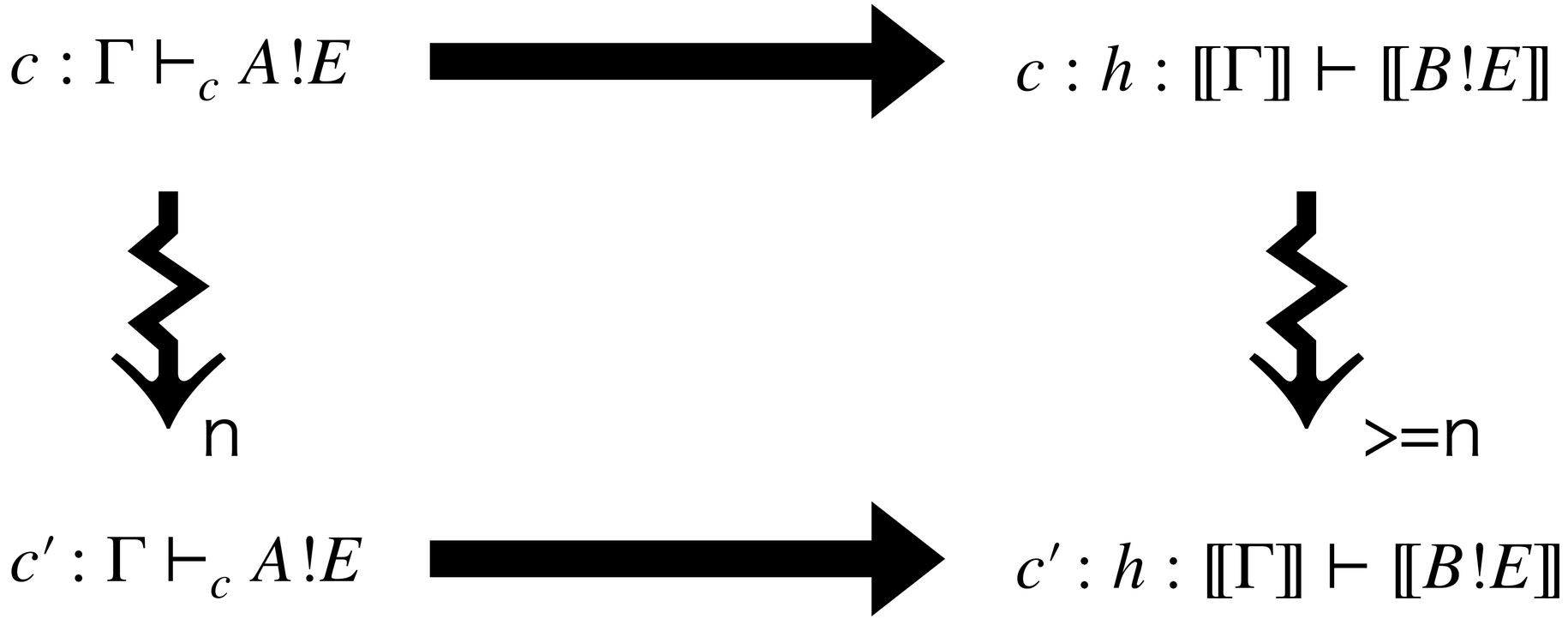
$[\![\Gamma]\!] \vdash c : h \rightsquigarrow^{+} v \equiv (c' : h) : B$ for some v

lemma: reduction preservation (WIP)

if $\Gamma \vdash c \rightsquigarrow c' : A\,!E, [\![\Gamma]\!] \vdash h : [\![A\,!E \Rightarrow B\,!E']\!]$ then

$[\![\Gamma]\!] \vdash c : h \rightsquigarrow^{+} c' : h : [\![B\,!E]\!]$

Normalization



Progress Lemma

lemma : Progress

if $\Gamma \vdash c : A\,!\,E$ then, either

$c = \mathsf{return}\ v$

$c = \mathsf{op}_i\ v\ y\,.\,c$

$c \to c'$ for some $\Gamma \vdash c' : A\,!\,E$

normal form

Normalization Algorithm

```
def nf(t : Γ ⊢ A) → Γ ⊢nf A :=
  case progress t
    | return v => return v
    | op v c => op v c
    | t ⇝ t' => nf t'
```

This algorithm always terminates

Future Work

- mechanization
  - lean? agda?
- extend source language
  - number (easier)
  - (type and effect) polymorphism
    - may or may not need bigger target language?

終わり

Appendix

value type A, B := bool | A -> $\underline{C}$ | $\underline{C}$ => $\underline{C}$

computation type $\underline{C}$, $\underline{D}$:= A!{op_1, op_2, ⋯, op_n}

Appendix

def: sn

if we say c is strong normalization,

$\forall c', (\Gamma \vdash c \to c') \Rightarrow c'$ is strong normalization

Appendix

$$
\begin{aligned}
[\![\mathsf{bool}]\!] &= \mathsf{bool} \\
[\![A \to B \,!\, E]\!] &= [\![A]\!] \to [\![B \,!\, E]\!] \\
[\![A \,!\, E \Rightarrow B \,!\, E']\!] &= [\![A \,!\, E \Rightarrow]\!][\![B \,!\, E']\!]
\end{aligned}
$$

$$
[\![A \,!\, E]\!] = \forall B \,.\, [\![A \,!\, E \Rightarrow]\!] B \to B
$$

$$
[\![A \,!\, E \Rightarrow]\!] B = \begin{cases} \mathsf{ret} : [\![A]\!] \to B \\ \mathsf{op}_i : [\![A_i]\!] \to \big([\![B_i]\!] \to B\big) \to B \quad \text{if } \mathsf{op}_i : A_i \to B_i \in E \end{cases}
$$