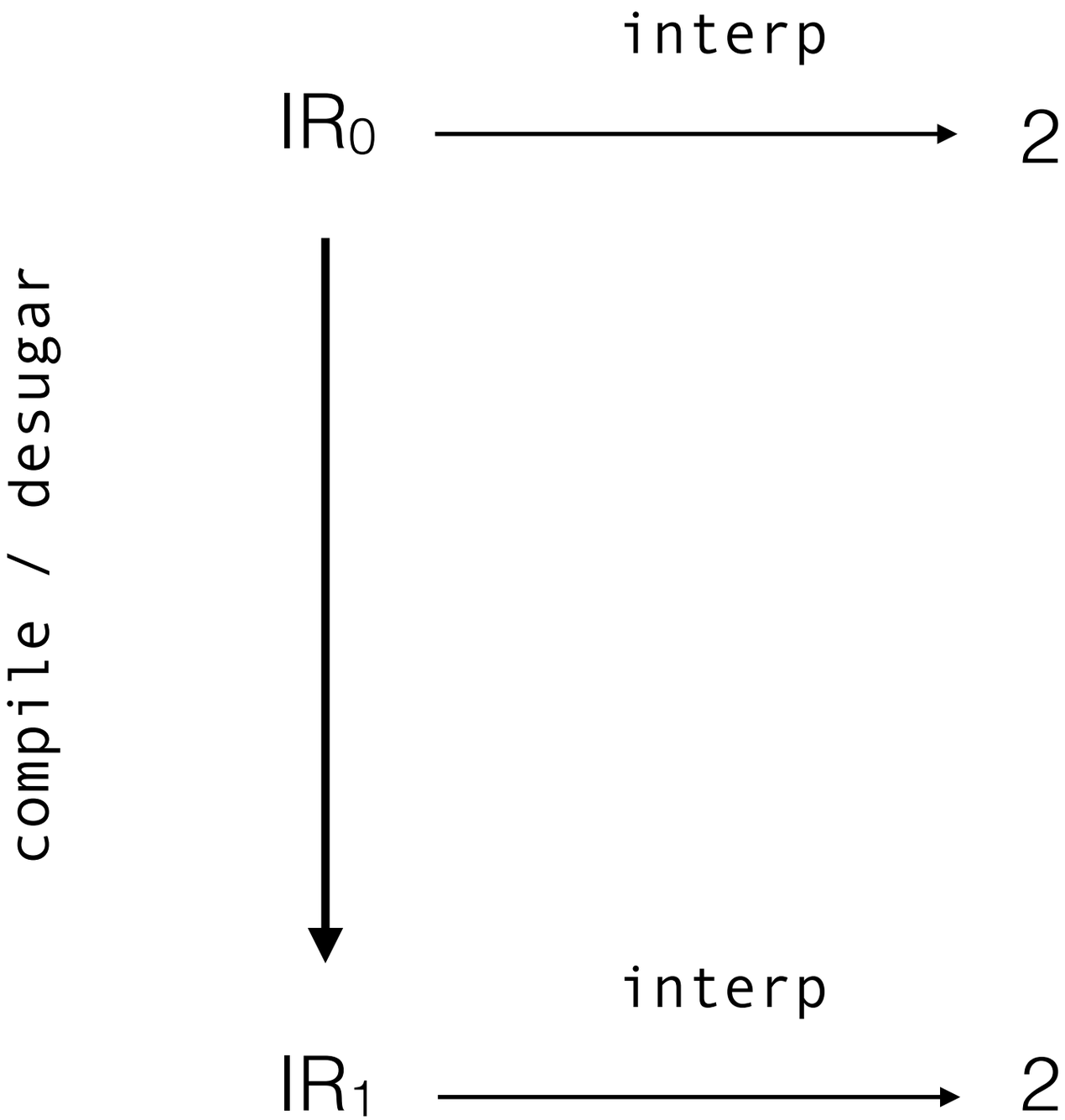
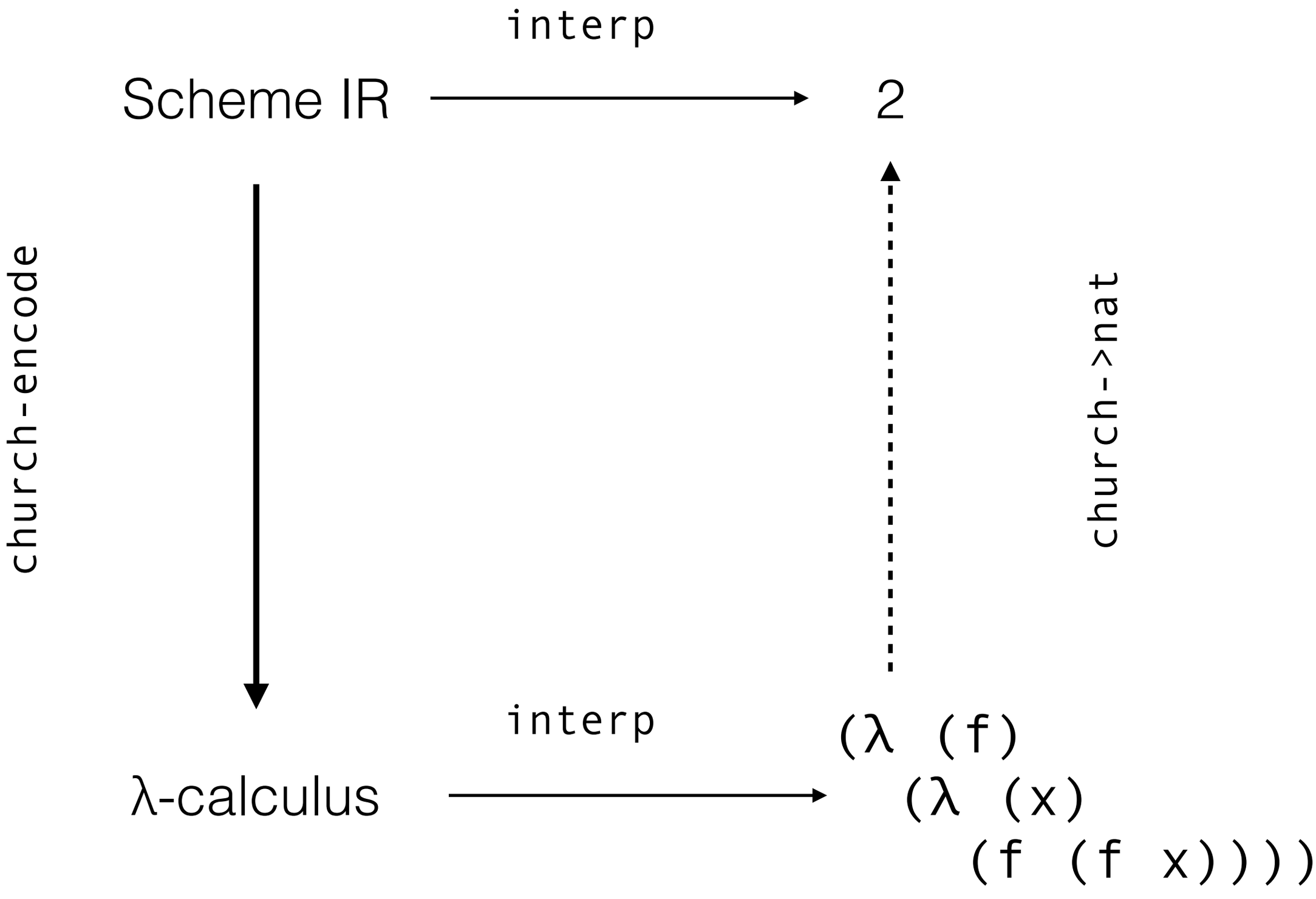
Semantics


interp
IR0
2
compile / desugar
II
interp
IR1
2

Assignment 1


interp
Scheme IR
2
church-encode
church->nat
interp
λ-calculus
(λ (f)
(λ (x)
(f (f x))))

Formal semantics

Axiomatic Semantics

Gives axioms for constructing sound proofs about programs (typically using Hoare logic).

Denotational Semantics

Provides a function that maps language forms into their denotations in a *known* domain.

Operational Semantics

Provides a step-by-step reduction of the program to a value in terms of program terms or an *abstract machine*.

```
((λ (f) (f (f (λ (x) x)))) (λ (x) x))
```

β

```
((λ (x) x) ((λ (x) x) (λ (x) x)))
```

β

```
((λ (x) x) (λ (x) x))
```

β

```
(λ (x) x)
```

$$\underbrace{((\lambda\ (x)\ E_0)\ E_1)}_{\text{redex}} \rightarrow_\beta E_0[x \leftarrow E_1]$$

Capture-avoiding substitution

$$E_0[x \leftarrow E_1]$$

$$\mathrm{FV}(\mathtt{x}) = \{\mathtt{x}\}$$

$$\mathrm{FV}(\mathtt{(\lambda\ (x)\ E_0)}) = \mathrm{FV}(\mathtt{E_0}) \setminus \{\mathtt{x}\}$$

$$\mathrm{FV}(\mathtt{(E_0\ E_1)}) = \mathrm{FV}(\mathtt{E_0}) \cup \mathrm{FV}(\mathtt{E_1})$$

$$x[x \leftarrow E] = E$$

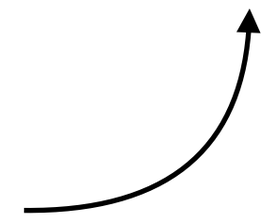
$$y[x \leftarrow E] = y \quad \text{where} \quad y \neq x$$

$$(E_0 \ E_1)[x \leftarrow E] = (E_0[x \leftarrow E] \ E_1[x \leftarrow E])$$

$$(\lambda \ (x) \ E_0)[x \leftarrow E] = (\lambda \ (x) \ E_0)$$

$$(\lambda \ (y) \ E_0)[x \leftarrow E] = (\lambda \ (y) \ E_0[x \leftarrow E])$$

where $y \neq x$ and $y \notin FV(E)$



β-reduction cannot occur when $y \in FV(E)$

α - renaming

```
(λ (x) (λ (y) x))
```

```
(λ (a) (λ (b) a))
```

α-renaming

$$(\lambda\ (x)\ E_0) \quad \rightarrow_\alpha \quad (\lambda\ (y)\ E_0[x \leftarrow y])$$

$$=_\alpha$$

η-reduction

$(\lambda\ (x)\ (E_0\ x)) \quad \rightarrow_{\eta} \quad E_0$ where $x \notin FV(E_0)$

Reduction

$$(\rightarrow) \;=\; (\rightarrow_\beta) \cup (\rightarrow_\alpha) \cup (\rightarrow_\eta)$$

$$(\rightarrow^*)$$

reflexive/transitive closure

Evaluation

E_0

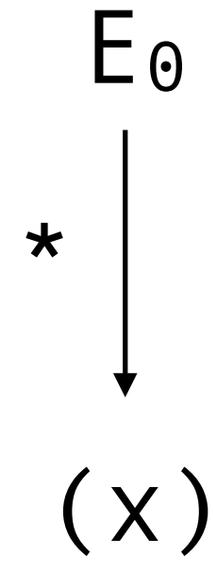
$\downarrow$ *

E_1

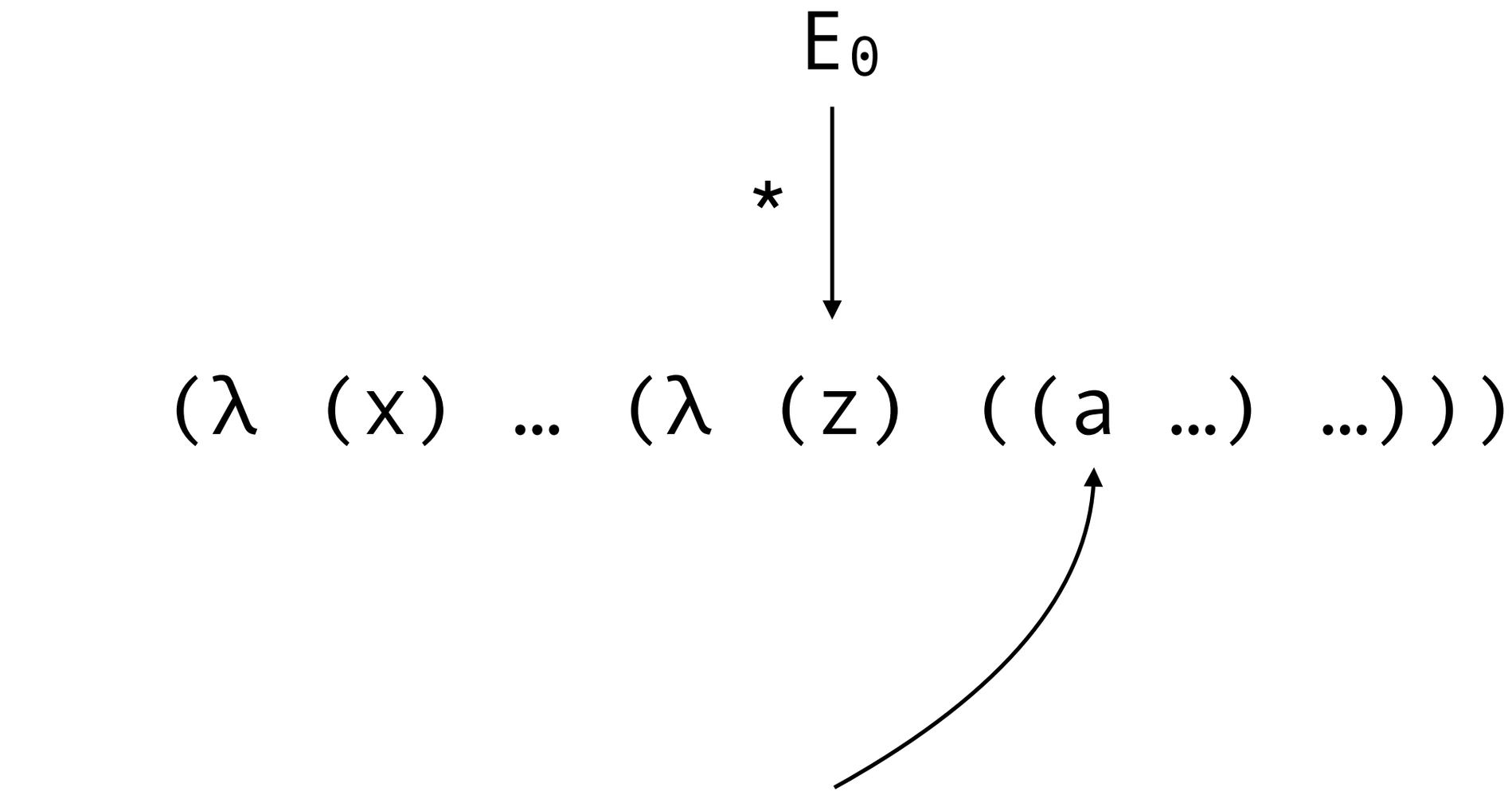
$\downarrow$ *

?

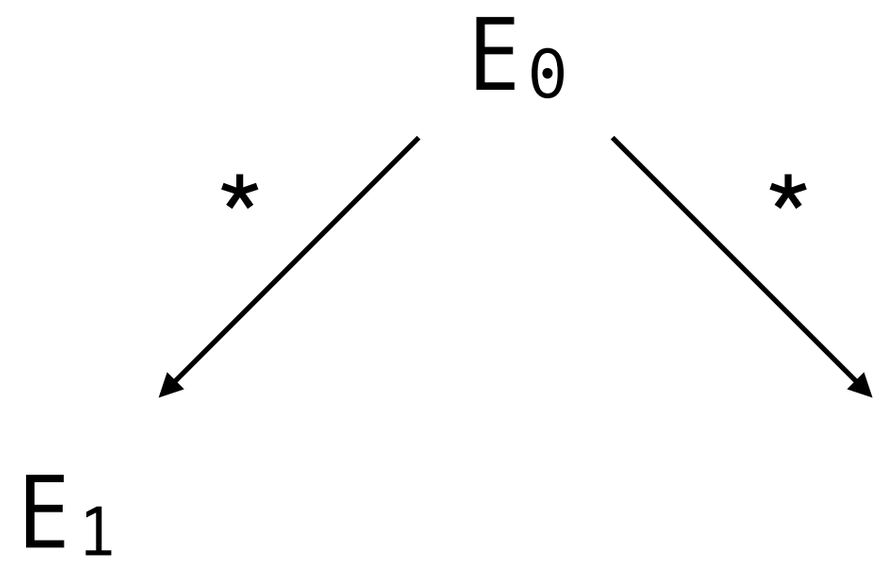
Evaluation to *normal form*



Evaluation to *normal form*



Evaluation Strategy



Evaluation Strategy

((λ (x) ((λ (y) y) x)) (λ (z) z))

→η ((λ (y) y) (λ (z) z))

→β (λ (z) z)

Evaluation Strategy

```
((λ (x) ((λ (y) y) x)) (λ (z) z))
```

→β ((λ (y) y) (λ (z) z))

→β (λ (z) z)

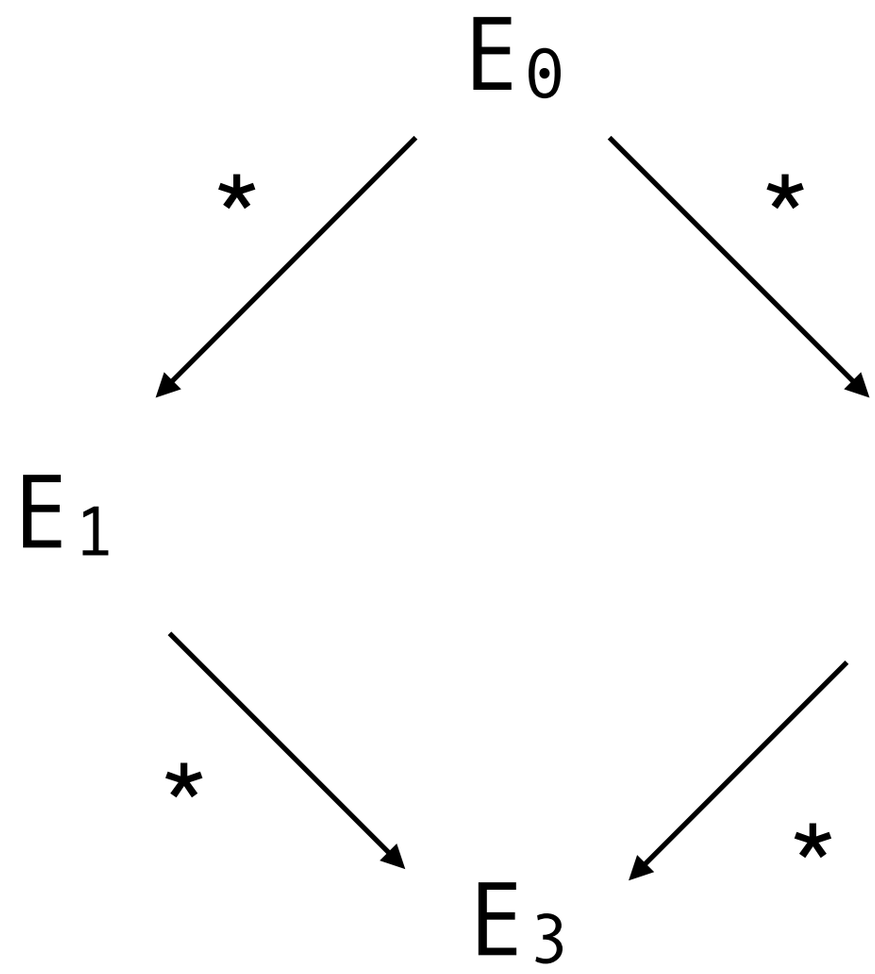
Evaluation Strategy

```
((λ (x) ((λ (y) y) x)) (λ (z) z))
```

$\rightarrow_\beta$ `((λ (x) x) (λ (z) z))`

$\rightarrow_\beta$ `(λ (z) z)`

Confluence



Church-Rosser Theorem

Applicative evaluation order

Always evaluates the *innermost* leftmost redex first.

Normal evaluation order

Always evaluates the *outermost* leftmost redex first.

Applicative evaluation order

```
((λ (x) ((λ (y) y) x)) (λ (z) z))
```

Normal evaluation order

```
(((λ (x) ((λ (y) y) x)) (λ (z) z)) (λ (w) w))
```

Call-by-value semantics

Applicative evaluation order, *but not under lambdas*.

Call-by-name semantics

Normal evaluation order, *but not under lambdas*.

Evaluation contexts

$$
\begin{array}{rcl}
v & ::= & (\lambda\ (x)\ e) \\
\mathscr{E} & ::= & (\mathscr{E}\ e) \\
 & | & (v\ \mathscr{E}) \\
 & | & \square
\end{array}
$$

$$
\begin{array}{rcl}
e & ::= & (\lambda\ (x)\ e) \\
 & | & (e\ e) \\
 & | & x
\end{array}
$$

Context and redex

$$\mathscr{E}[\overbrace{(\mathsf{v}\ \ \mathsf{v})}^{\mathsf{r}}] =$$

```
((((λ (x) ((λ (y) y) x)) (λ (z) z)) (λ (w) w))
```

$\mathscr{E}$ = (□ (λ (w) w))

r = ((λ (x) ((λ (y) y) x)) (λ (z) z))

Context and redex

$\mathscr{E}$[r] =

((λ (x) ((λ (y) y) x)) (λ (z) z)) (λ (w) w))

$\mathscr{E}$ = (□ (λ (w) w))

r = ((λ (x) ((λ (y) y) x)) (λ (z) z))

→β ((λ (y) y) (λ (z) z))

Put it back together:

$\mathscr{E}$ = (□ (λ (w) w))

r = ((λ (x) ((λ (y) y) x)) (λ (z) z))

→β ((λ (y) y) (λ (z) z))

↓ $\mathscr{E}$[r]

(((λ (y) y) (λ (z) z)) (λ (w) w))

Some exercises

```
1)   (((λ (y) y) (λ (z) z)) (λ (w) w))
```

```
2)   ((λ (u) (u u)) (λ (x) (λ (x) x)))
```

```
3)   (((λ (x) x) (λ (y) y))
       ((λ (u) (u u)) (λ (z) (z z))))
```