

## Lecture 23

Recall Call-by-name  
 $\lambda$ -calculus

Lazy Functional Languages  
(Haskell, Miranda)

motivation  $(\lambda x. (\lambda y. y)) \Omega = \lambda y. y$

and not an  $\omega$ -computation

- "Referential Transparency".

In terms of the  $\lambda$ -calculus

$$(p) \quad ((\lambda x. e_1) e_2) \rightarrow_p e_1[e_2/x]$$

$$(\lambda x. e_1) e_2 \rightarrow e_1$$

$$(op) \quad \frac{e_1 \rightarrow_{\beta} e_1'}{e_1 e_2 \rightarrow_{\beta} e_1' e_2}$$

$$(arg) \quad \frac{e_2 \rightarrow_{\beta} e_2'}{e_1 e_2 \rightarrow_{\beta} e_1 e_2'}$$

$$(\xi) \quad \frac{e \rightarrow_{\beta} e'}{\lambda x. e \rightarrow_{\beta} \lambda x. e'}$$

Note: CBN does not use  
(arg) and ( $\xi$ ) rules.

In terms of Contexts and Reduction

$$\mathcal{C}[] ::= []$$

$$| \mathcal{C}[] e$$

$$| \mathcal{C}[\lambda x. e]$$

$$\begin{array}{c} | \text{ } e \text{ } \dots \text{ } | \\ | \lambda x. e \text{ } [ \text{ } ] \end{array}$$

CBN does not use contexts  $e \text{ } [ \text{ } ]$   
 $\neg(\text{arg})$ , and  $\lambda x. e \text{ } [ \text{ } ]$  ( $\S$ )

$$e \text{ } [ (\lambda x. e_1) e_2 ] \rightarrow_{\beta} e \text{ } [ e_1 [e_2/x] ]$$

where

$$e \text{ } [ \text{ } ] ::= [ \text{ } ] \\ | e \text{ } [ \text{ } ] e$$

is a CBN Evaluation  
Context.

Any evaluation context  $e \text{ } [ \text{ } ]$   
 can be represented as a stack  
 of Basic Contexts

$$B[ \text{ } ] ::= [ \text{ } ] \mid [ \text{ } ] e$$

Build  $e \text{ } [ \text{ } ]$  by placing

the BE ] as "Russian Matryona Dolls".

## Evaluation Strategy

- Find a redex in left "spine"  
(left-most, outermost)
- Log the context  $C[ ]$
- Reduce the redex,  
keeping context intact
- \* If no redex found, we are  
in cbn-nf. (WHNF)

## Weak Head Normal Form

$$e ::= \boxed{x} \mid \lambda x. e \mid \dots$$

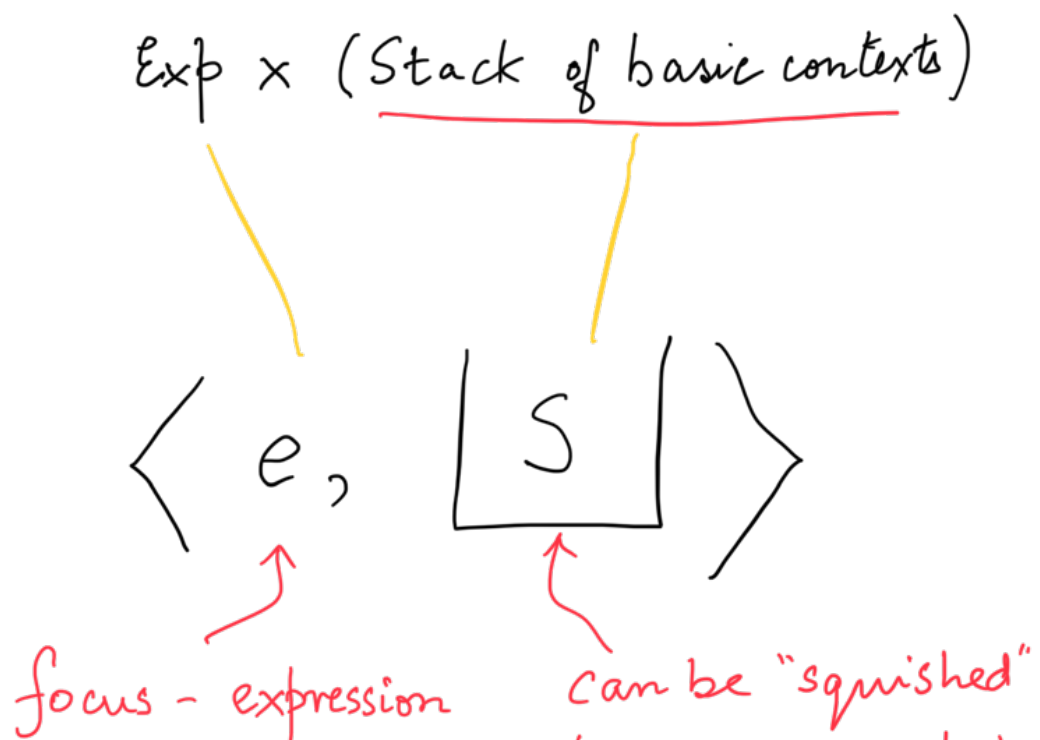
$$\boxed{x \ e_1 \dots e_n} \quad (n \geq 1)$$

$x \ e_1 \dots e_n \ (n \geq 0)$   
(unevaluated)

## CONTEXT-REDUCTION "MACHINE"

MACHINE in the sense that it is an algorithm ... rewriting a machine configuration one step at a time

Configurations:



to yield a context  
 $e[ ]$

- "stack up" basic contexts

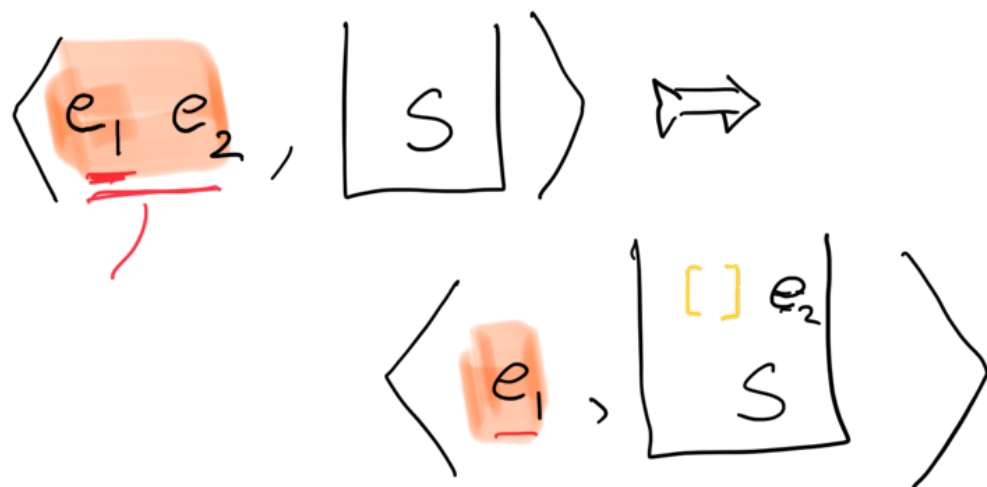
$B[ ] ::= [ ]$  — hole

$| [ ] e$

hole in  
op-position

argument

(Decompose) — Looking for redex



"Shift focus to op-position  $e_1$ "

• push context  $[ ] e_2$  onto stack

$$\mathcal{C}[e_1 e_2] \rightsquigarrow \mathcal{C}[\underbrace{[e_1]} e_2]$$

(apply) - found a redex, now reduce.

$$\langle \lambda x.e_1, \begin{array}{|c|} \hline [ ] e_2 \\ \hline S \\ \hline \end{array} \rangle \Rightarrow$$

$$\langle e_1[e_2/x], \begin{array}{|c|} \hline S \\ \hline \end{array} \rangle$$

- Pop the basic context  $[ ] e_2$  off stack

- Replace focus by "contractum"

$$e_1[e_2/x]$$

$$\mathcal{C}[\underbrace{[\lambda x.e_1]} e_2] \rightarrow \mathcal{C}[e_1[e_2/x]]$$

Note: — the "decompose" step is only analysing the context, and  
 — "apply" is doing the real computation

"COLLAPSING" a Stack of Basic Contexts

$$B[\ ] ::= [\ ] \mid [\ ] e$$

into an evaluation context  $\mathcal{E}[\ ]$

$$\text{Let coll} \left( \boxed{\phantom{\quad}} \right) = [\ ]$$

↑  
empty stack

↑  
context  
that is only  
a "hole"

$$\text{coll} \left( \boxed{[\ ]^e \mid s} \right) =$$

$$\text{let } \varphi \vdash \gamma = \text{coll}(|s|)$$



= ...

$$\stackrel{\text{in}}{\mathcal{C}}[[\ ]e]$$

Now, at any point in the execution  
the expression corresponding  
to machine configuration

$$\langle e, [s] \rangle$$

represents  
the expression

$$\mathcal{C}[e]$$

where  $\mathcal{C}[\ ] = \text{coll}([s])$

### NOTE

Since stacked basic closures are always  
of the form  $[\ ]e$ , we can just  
obtain the operator

stack up  $\hookleftarrow$ , leaving me up-  
position hole  $[ ]$  implicit.

## CLOSURES & CLOSURE MACHINES

We know the  $e_1[e_2/x]$  is  
an expensive operation...

So we build "closures"

- expression (with variables)
- +  
- table (to find bindings  
for "apparently free"  
but actually bound  
variables, which

have to be  
substituted for )

Data structure:

$$\text{Closure} \triangleq \text{Exp} \times \text{Table}$$

and

$$\gamma \in \text{Table} \triangleq X \xrightarrow{\text{fn}} \text{Closure}$$

(mutually recursive)

Well-founded, since  $e \in \text{Exp}$   
- need not have a free variable  
- and  $\gamma \in \text{Table}$  can be empty

$\text{VClosure}$

$\text{Let } \text{val} ::= (\lambda x. e, \gamma)$

value

table

$$| (x e_1 \dots e_n, \gamma)$$

if you really insist

Closure - Semantics for CBN evaluation

$\Rightarrow_n \subseteq \text{Closure} \times \text{VClosure}$   
 Input - Output Spec

$$\text{(var)} \quad \frac{\gamma(x) \Rightarrow_n \text{val}}{(x, \gamma) \Rightarrow_n \text{val}}$$

$$\text{(abs)} \quad \frac{}{(\lambda x. e, \gamma) \Rightarrow_n (\lambda x. e, \gamma)}$$

$$\text{(app)} \quad \frac{\begin{array}{l} (e_1, \gamma) \Rightarrow_n (\lambda x. e', \gamma') \\ (e', \gamma' [x \mapsto (e_2, \gamma)]) \Rightarrow_n \text{val} \end{array}}{(e_1 e_2, \gamma) \Rightarrow_n \text{val}}$$

(var):

- If  $x \in \text{dom}(\gamma)$ , look up the table for the closure corresponding to  $x$  and evaluate it to an answer.  
 (val)

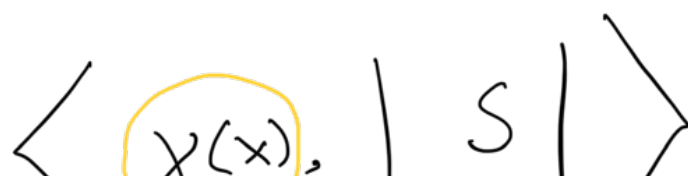
(abs)

- An abstraction closure is an answer already (0 steps)

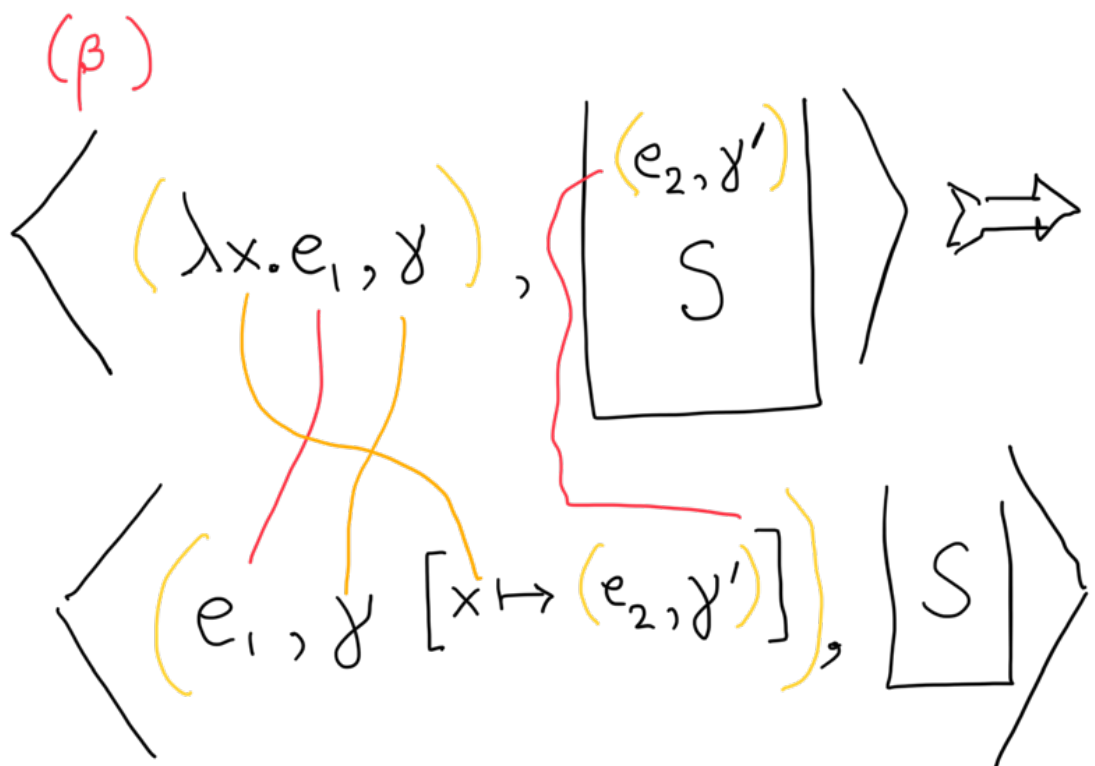
(app) . CBN function call .  $e_1 e_2$

- First simplify the closure  $(e_1, \gamma)$  corresponding to  $op$   $e_1$  to an abstraction closure  $(\lambda x. e', \gamma')$
- Then evaluate the body  $e'$  wrt  $\gamma'$  extended by binding formal parameter  $x$  to the argument closure  $(e_2, \gamma)$
- Return the result  $val$

## COMBINING THE CLOSURE REPRESENTATION WITH THE CONTEXT M/C

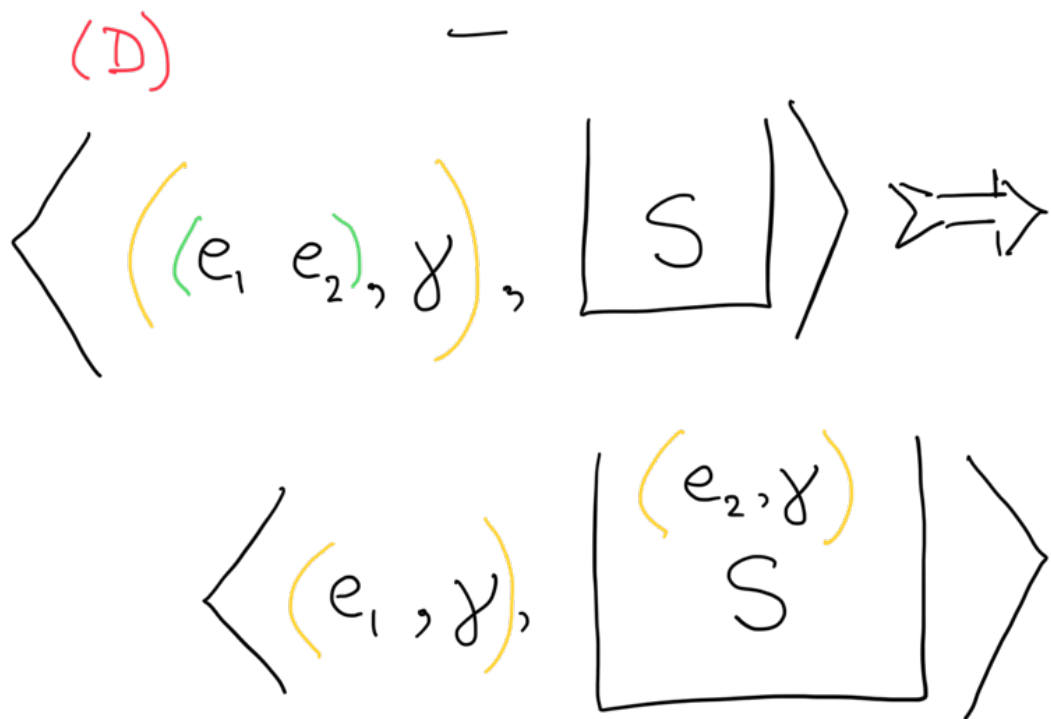


- Look up variable  $x$  in table  $\gamma$ , and now focus on that closure for further reduction.



Focus now on body  $e_1$  of the abstraction  
 extending the table  $\gamma$  by binding formal  
 parameter  $x$  to the argument closure  
 $(e_2, \gamma')$  which is popped off the stack.

(first step of a function call)



Shift focus from  $e_1, e_2$  to op-position  $e_1$ ,  
pushing arg closure  $(e_2, \gamma)$  on stack

---

When does this machine  
(KRIVINE Machine) stop?

— when we have a config

$\gamma \mid S \mid$

$\Rightarrow \langle (x, \gamma), \sqcup \rangle /$   
 but  $x \notin \text{dom}(\gamma)$ .

OR  
 $\langle (\lambda x. e, \gamma), \sqcup \rangle$   
 empty stack.

i.e when "unloading" the  
 machine gives an  
 expression in Weak HNF.

---

Call-by-value

(B<sub>v</sub>)  $\frac{}{\lambda x. e \rightarrow \dots}$



$$\vdash (\lambda x. e_1) v \rightarrow_{\beta} e_1[v/x]$$

$v$  - a value.

$$(op) \quad \frac{e_1 \rightarrow_{\beta} e_1'}{e_1 e_2 \rightarrow_{\beta} e_1' e_2}$$

$$(arg) \quad \frac{e_2 \rightarrow_{\beta} e'}{v e_2 \rightarrow_{\beta} v e'}$$

(No  $(\xi)$  rule)

Evaluation Contexts for CBV

$$\begin{array}{l} \mathcal{E}[ \ ] ::= [ \ ] \\ \quad | \mathcal{E}[ \ ] e \\ \quad | v \mathcal{E}[ \ ] \end{array}$$

$$v ::= \lambda x. e \quad | \quad x \ v_1 \dots v_n$$

if you really  
insist...

Context Search is for left-most  
inner-most (not below  $\lambda$ )  
redex.

Context Machine for CBV.

Basic Contexts:

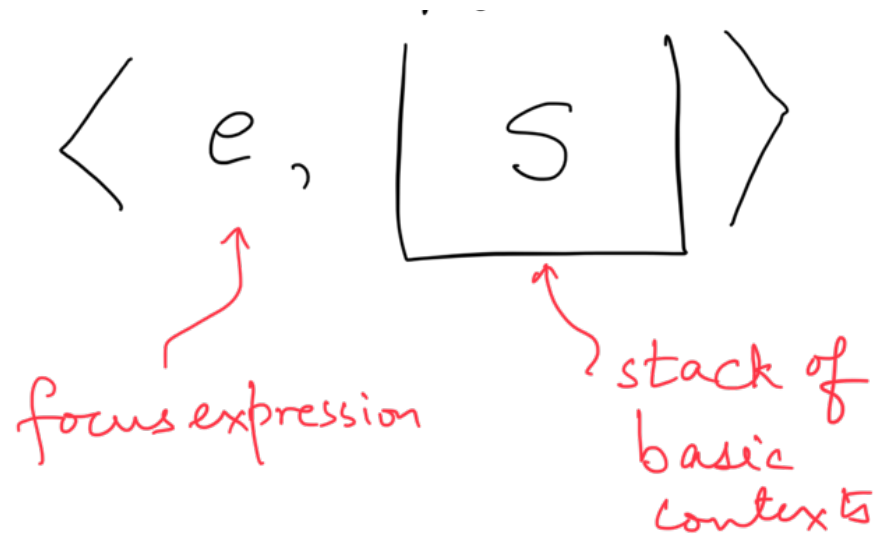
$$\begin{aligned} B_{\check{v}}[ ] &::= [ ] e \\ &| v [ ] \\ &| [ ] \end{aligned}$$

Any evaluation context

$E[ ]$  is obtained by

"collapsing in" a stack of Basic Contexts.

Machine Configurations:



Note:  $\underbrace{[\ ]}_{\text{empty stack}} = \underbrace{[ \ ]}_{\text{hole}}$

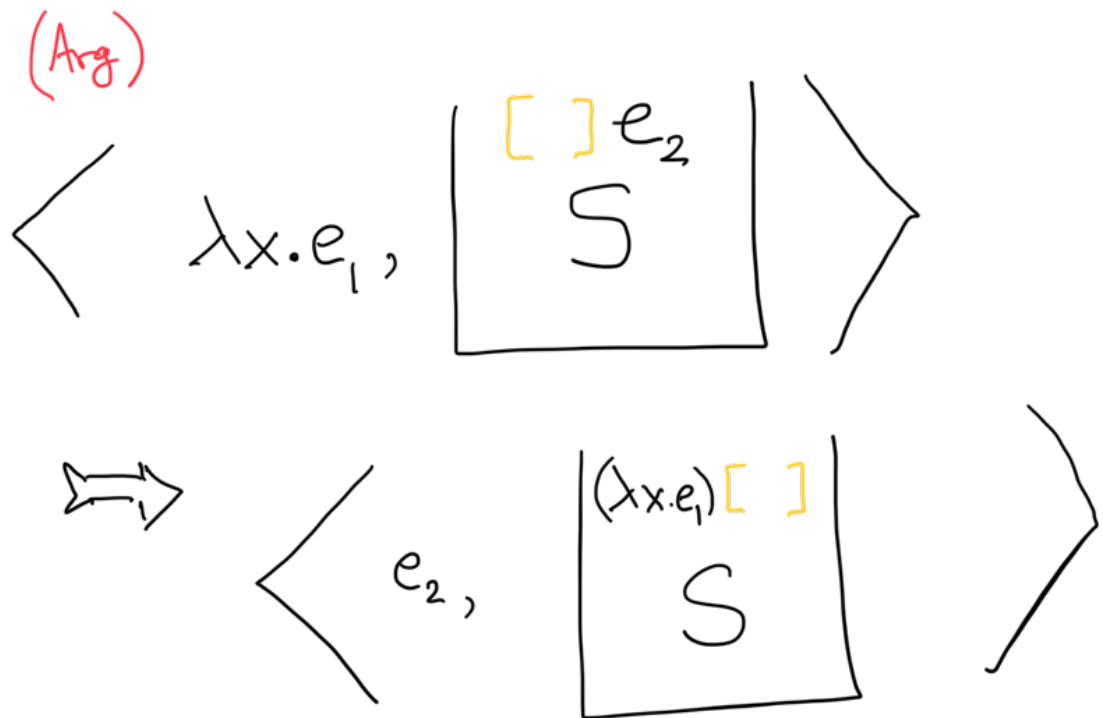
(Decompose)

$$\langle e_1 e_2, [S] \rangle \Rightarrow$$

$$\langle e_1, [ [ ] e_2 S ] \rangle$$


---

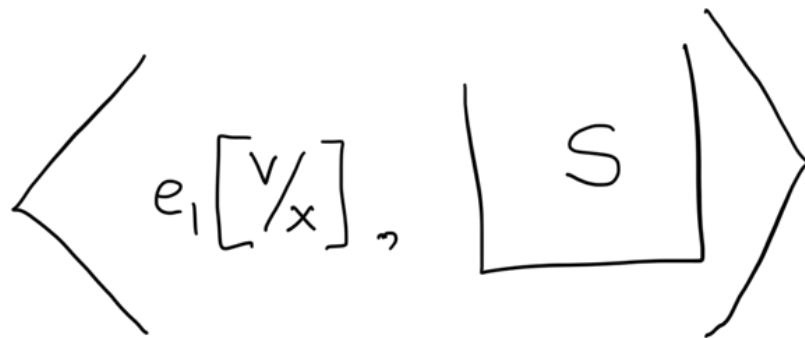
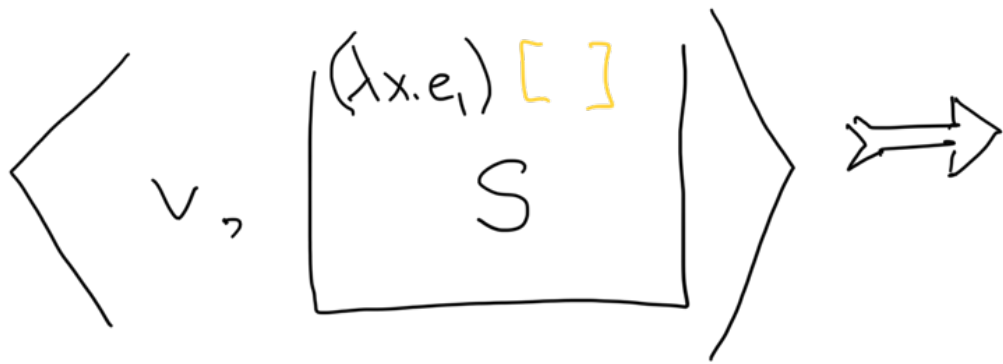
As before



Note: Having found an abstraction in the op-position, swap contexts on the top-of-stack, to now focus on the arg position

$$c[[\lambda x.e_1] e_2] \rightsquigarrow c[\lambda x.e_1 [e_2]]$$

(Apply)



$$e [ (\lambda x. e_1) [v] ] \xrightarrow{\beta_r} e [ e_1 [v/x] ]$$

Again: at any point in the execution, configuration



corresponds to expression

$\mathcal{E}[e]$  where

$$\mathcal{E}[\ ] = \text{coll}(\lfloor S \rfloor)$$

(for a slightly extended  
definition of coll)

Exercise: Extend the  
definition of coll.

Closures for call-by-value

$$\text{Table} \triangleq X \xrightarrow{\text{fin}} V_{\text{Closure}}$$

$$vcl \in V_{\text{Closure}} \triangleq \left\{ (\lambda x. e, \gamma) \mid e \in \mathcal{E}_p \right.$$

$\tau \quad \tau \quad \tau$

and  $y \in \text{table}$  }

## Cbr rules with closures

$$(Var) \frac{}{(x, y) \Rightarrow_v \underbrace{y(x)}_{\text{a value closure}}}$$

$$(abs) \frac{}{(\lambda x.e, y) \Rightarrow_v (\lambda x.e, y)}$$

$$(app) \frac{\begin{array}{l} (e_1, y) \Rightarrow_v (\lambda x.e', y') \\ (e_2, y) \Rightarrow_v val_2 \\ (e', y'[x \mapsto val_2]) \Rightarrow_v val_3 \end{array}}{(e_1 e_2, y) \Rightarrow_v val_3}$$

(var) . Look up table  $y$  for  $x$  -  
return the value closure  $y(x)$

(abs) . Already a value closure

... . First evaluate  $e$ . wrt  $y$  to

(app)

value closure  $(\lambda x. e', \gamma')$

- Then evaluate  $e_2$  wrt  $\gamma$  to value closure  $val_2$
- Finally evaluate body of abstraction, i.e.,  $e'$  wrt  $\gamma'$  extended by  $x$  bound to  $val_2$
- and return the result  $val_3$