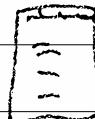


(-1)

# Goal: Implement a VM

what does a program mean?

"1 + 1" means "2"

"word doc" means 

a semantics - english (human)

- math (universal)

- code

$$\forall x, y \in \mathbb{N}: x + y = y + x$$

BNF

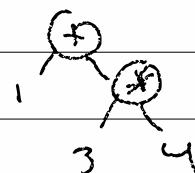
$\mathcal{J}_0$ :  $e := v \mid (+ e e) \mid (* e e)$  or  
 $v := \text{number}$

"(+ 1 (\* 3 4))"  $\in \mathcal{J}_0$ . e

"(+ 1 ))"

$e := v \text{ or } \begin{array}{c} + \\ e \quad e \end{array} \text{ or } \begin{array}{c} * \\ e \quad e \end{array}$

$v := \text{number}$



1-2  = new Add (new Num(1),  
new Mult(

class E { }  
class Add : E { }

new Mult(

class Num : E { }  
new Num(3),

class Mult : E { }  
new Num(4)) )

Add ::= Add ( E \* 1 , E \* n ) {

this, l = 1; this, n = n; }

( + 1 (\* 3 4) )

Semantics = meaning of programs

interpreter : a program in ~~the~~ language M  
that tells you the semantics of  
language O

Virtual machine : a fast interpreter we like

interp ( w big step semantics ) : e → v

interp v = v

interp ( + e<sub>1</sub> e<sub>2</sub> ) = (interp e<sub>1</sub>) +<sub>v</sub> (interp e<sub>2</sub>)

+<sub>v</sub> (Num n<sub>1</sub>) (Num n<sub>2</sub>) = Num (n<sub>1</sub> + n<sub>2</sub>)

Num\* = V\*

1-3)       $\Leftarrow$        $\Rightarrow$

virtual E::interp() = 0;  
 // interp v = v

E\* Num::interp() { return this; }

// interp (+ e<sub>1</sub> e<sub>2</sub>) = (interp e<sub>1</sub>) + v (interp e<sub>2</sub>)

E\* Add::interp() {  
 Num n<sub>1</sub> = this.l.interp();  
 Num n<sub>2</sub> = this.r.interp();  
 return new Num(n<sub>1</sub> + n<sub>2</sub>); }

---

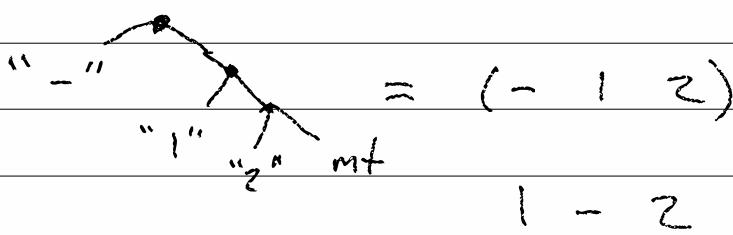
$$(- e_1) = (* -1 e_1)$$

$$(- e_1 e_2) = (+ e_1 (- e_2))$$

desugar (expander) [compiler] : string tree  
 string expr

Sexpr = string or pair of sexpr  
 or empty

$\rightarrow S_{\alpha}$



2-1)

Sexpr

$J_0$

$$\text{desugar } (- e_1) = (* \ -1 \ (\text{desugar } e_1))$$

$$\begin{array}{c} - \\ \diagup \quad \diagdown \\ e_1 \quad e_2 \end{array} \text{mt} \quad (- e_1 \ e_2) = (+ \ (\underline{d} \ e_1) \ (\cancel{*} \ (- e_2))) \\ (+) = 0$$

$$\begin{array}{c} + \\ \diagup \quad \diagdown \\ e_1 \quad \text{more} \end{array} \quad (+ \ e_1 \ . \text{more}) = (+ \ (\underline{d} \ e_1) \ (\underline{d} \ (+ \ . \text{more}))) \\ (+) = 1$$

$$(* \ e_1 \ . \text{more}) = (* \ (\underline{d} \ e_1) \ (\underline{d} \ (* \ . \text{more})))$$

$$J_1 \quad e := v \mid (\text{if } e_1 \ e_2 \ e_3) \mid (e \ e \dots)$$

$$v := \text{number} \mid \text{bool} \mid \text{prim}$$

$$\text{prim} := + \mid * \mid / \mid - \mid \leq \mid < \mid = \mid > \mid \geq$$

$$\text{interp } v = v$$

$$\text{interp } (\text{if } e_1 \ e_2 \ e_3) =$$

$$c = \text{interp } e_1$$

$$e_k = \underline{\text{if}} \ c \ \underline{\text{et}} \ \underline{o.v} \ \underline{\text{ef}}$$

$$\text{return}_m \ \text{interp } e_k$$

$$\text{interp } (e_1 \ e_2 \ \dots \ e_n) =$$

$$p = \text{interp } e_1 \ (\text{must be a prim})$$

$$v_0 \dots v_n = \text{interp } v_0 \ \dots \ \text{interp } v_n$$

$$\text{ref } S(p, v_0 \dots v_n)$$

2-2/

$\text{vs delta}(\text{prim } p, \overset{\text{high}}{v^*} \text{ vs}) =$

if ( $p == \text{ADD}$ )

return new Num( $\text{vs}[0].n + \text{vs}[1].n$ )

if ( $p == \text{LT}$ )

return new Bool( $\text{vs}[0].n < \text{vs}[1].n$ )

big-step has a big problem

:  $e \Rightarrow v$

- it is partial

- it says nothing about "in between"

- very un-math-like and clumsy

- inefficient / unhelpful for implementation

small step :  $e \Rightarrow e'$

step (if true  $e_1$   $e_2$ ) =  $e_1$

step (if false  $e_1$   $e_2$ ) =  $e_2$

step ( $P v_0 \dots v_n$ ) =  $\delta(P, v_0 \dots v_n)$

if step  $e_c = e'_c$  then step (if  $e_c$   $e_1$   $e_2$ )  
= (if  $e'_c$   $e_1$   $e_2$ )

if step  $e_i = e'_i$  then step ( $e_0 \dots e_i e_{i+1} \dots e_n$ )  
= ( $e_0 \dots e'_i e_{i+1} \dots e_n$ )

$$\begin{array}{c}
 \overbrace{(+) + (2+3)}^{2-3} = 2 + \overbrace{(2+3)}^{\text{step}} = 2+5 = 7 \\
 = (1+1) + 5
 \end{array}$$

In context, C = hole

- | if C e e
- | if e C e
- | if e e C
- | (e ... C e ...)

$$\text{plug} : C \times e \rightarrow e$$

$$\text{plug hole } e = e \quad (\text{plug } C e_p)$$

$$\text{plug (if } C e_1 e_2) e_p = (\text{if } e_p e_1 e_2)$$

$$\text{plug } (e_b \dots C e_n \dots) e_p = (e_b \dots (\text{plug } C e_p) e_n \dots)$$

$$\begin{array}{lcl}
 \text{plug } ((+) + (2+3)) & (1+1) = (1+1) + (2+3) \\
 \tilde{C} & 2 = 2 + (2+3)
 \end{array}$$

$$\text{Parse} : e \Rightarrow C \times e$$

2.4/

$e \rightarrow e$

step  $C[\text{if true } e \text{ else } e_f] = C[e]$

step  $C[\text{if false } e \text{ else } e_f] = C[e_f]$

step  $C[p \rightarrow v_0 \dots v_n] = C[\delta(p, v_0 \dots v_n)]$

find-redex :  $e \rightarrow C \star e$

$\uparrow$   
redex

reducible expression

---

When do two programs mean the same thing?

" $1 + 2$ "      " $2 + 1$ "

" $4 \text{ billion} + 1 + 2$ "      " $2 + 1 + 4 \text{ billion}$ "

$\vdash C = \mathbb{B}$

"quicksort"      "mergesort"      "heapsort"

"insertionsort"

$\forall i, \text{ "mergesort"} + i = \text{"hs"} i$

introduction      merge       $\vdash C = (\mathbb{B} L)$

$\forall C, C[x] = C[y] \rightarrow \text{observing } (x=y)$   
equivalence

time  $e = \text{days} \times \text{secs}$

3-1/ step :  $e \Rightarrow e \rightarrow \rightarrow \rightarrow v$

$C[\text{if true } e_1 \text{ or } e_2] \rightarrow C[e_1]$

$\dots + \dots * \dots - \dots (1+1) \dots * \dots +$

$C := \text{hole} \mid \text{if } C e_1 e_2 \mid \text{if } e_1 C e_2 \mid \text{if } e_1 e_2 C$   
 $(e_1 \dots C e_2 \dots)$

evaluation context, E

$E := \text{hole} \mid \text{if } E e_1 \mid (v \dots E \dots)$

$E_{\text{hole}}$

$E_{\text{if}}$

$E_{\text{app}}$

$E[\text{if true } e_1 \text{ or } e_2] \rightarrow E[e_1]$

$E[\text{if false } e_1 \text{ or } e_2] \rightarrow E[e_2]$

$E[(p \ v_0 \dots v_n)] \rightarrow E[\delta(p, v_0 \dots v_n)]$

find-redex :  $e \rightarrow (E, e)$  or  $\#\text{false}$

find-redex  $v = \text{false}$

$\text{fr } (\text{if } e_0 e_1 e_2) = \text{case } (\text{fr } e_0) \text{ with}$

$\#\text{false} \Rightarrow (\text{hole}, (\text{if } e_0 e_1 e_2))$

$(E, e_0) \Rightarrow (\text{if } E e_1 e_2, e_0)$

3-2/

$$fr(v \dots e_0 e_1 \dots e_n) =$$

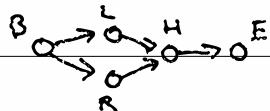
$$(E, e'_0) = fr e_0$$

$$( (v \dots E e_1 \dots e_n), e'_0 )$$

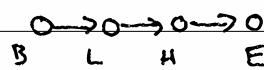
new EApp(v, E, e\_1 \dots e\_n)

$$fr(v \dots) = (\text{hole}, (v \dots))$$

step c



skip<sub>E</sub>



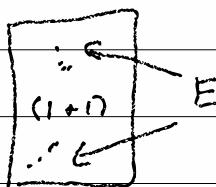
The Standard Reduction theorem

Alonzo

Church

(Curry-Rosser)

Rosser



$$E = (\text{if } (* \neq 8 (+ 1 2 (* \dots \dots \dots \dots)))$$

$$\dots \dots \text{ (if } \dots \dots \text{ hole } \dots \dots \text{)})^{(434)})$$

$$E[(1+i)] \rightarrow E[2]$$

$\underbrace{\dots}_{\text{in time}}$

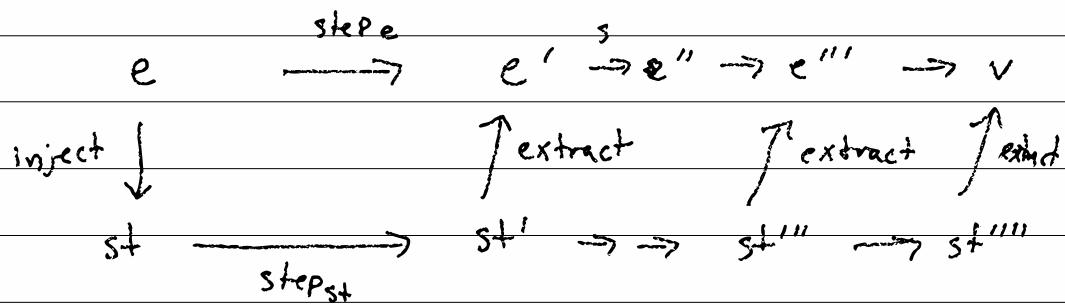
in space

$\underbrace{\dots}_{\text{in time}}$

in space

3-3)  $\text{step}_E = e \rightarrow e$

machine semantics



$$CC_0 \quad st = \langle e, E \rangle$$

$$\text{inject } e = \langle e, \text{hole} \rangle$$

$$\text{extract } \langle e, E \rangle = E[e]$$

$$\text{done? } \langle v, \text{hole} \rangle = \text{true}$$

- 1  $\langle \text{if } e_c \text{ et } e_f, E \rangle \Rightarrow \langle e_c, E[\text{if hole et } e_f] \rangle$
  - 2  $\langle \text{true}, E[\text{if hole et } e_f] \rangle \Rightarrow \langle e_f, E \rangle$
  - 3  $\langle \text{false}, E[\text{if hole et } e_f] \rangle \Rightarrow \langle e_f, E \rangle$
  - 4  $\langle e_0 e_1 \dots e_n, E \rangle \Rightarrow \langle e_0, E[(\text{hole } e_1 \dots e_n)] \rangle$
  - 5  $\langle v, E[v_0 \dots \text{hole } e_0 e_1 \dots] \rangle \Rightarrow \langle e_0, E[(v_0 \dots v \text{ hole } e_1 \dots)] \rangle$
  - 6  $\langle v_n, E[p \dots \text{hole}] \rangle \Rightarrow \langle \delta(p, v_0 \dots v_n), E \rangle$
- $(y_n, z_n) \rightarrow (l_n, l_n)$

3-4)

HL: ( $\text{test} \ ' (+ 1 (* 2 (\text{if true } 3 4)))$ )  
7)

$\text{test} ( \text{new SExpr} ( \text{new Atom} ("+"),$   
~~(new Se( new A ("1"),~~  
.....,  
,  
 $\text{new Num} (7))$

$\text{test} ( \text{se}, \text{ex-val}) =$

$e = \text{desugar se}$

$\text{big-step eval } (e) = \text{actual-big-eval}$

$\text{if } (\text{abv} \neq \text{ev}) \{ \text{error} \}$

$\text{small-step eval } (e) = \text{acc-sm-eval}$

$\text{if } (\text{asv} \neq \text{ev}) \{ \text{error} \}$

$\text{cc-eval } e = \text{accv}$

$\text{if } (\text{accv} \neq \text{ev}) \{ \text{error} \}$

$\text{ll-eval } e = \text{allv}$

$\text{if } (\text{allv} \neq \text{ev}) \{ \text{error} \}$

3-5/

low level - eval e =

print e as C constructors into "x.c"

compile "x.c" into "x.bin"

run "x.bin" and capture output

parse output

return value

cc x.c ll.c -o x.bin

$$q-1) \quad C(\_o \rightarrow CK_o$$

$$st = \langle e, k \rangle$$

$$k = k_{ret} \quad // \text{ hole}$$

continuation       $kif\ e\ e\ k \quad // \text{ if } E \times e$

Kontinuation       $kapp(v...) (e...) k \quad // (v... E e...)$

stack k

$$A \rightarrow kapp(v...) (e...) B$$

$$\hookrightarrow B[v \dots A e \dots]$$

$$\text{inject } e = \langle e, k_{ret} \rangle$$

$$\text{extract } \langle e, k \rangle = k_{\text{intoE}}(k)[e]$$

$$\text{done? } \langle v, k_{ret} \rangle = \text{the}$$

$$\langle \text{if } e_c\ e_t\ e_f, k \rangle \mapsto \langle e_c, kif(e_t, e_f, k) \rangle$$

$$\langle \text{true}, kif(e_t, e_f, k) \rangle \mapsto \langle e_t, k \rangle$$

$$\langle \text{false}, kif(e_t, e_f, k) \rangle \mapsto \langle e_f, k \rangle$$

$$\langle (e_0\ e_1\dots), k \rangle \mapsto \langle e_0, kapp(\(), (e_1\dots), k) \rangle$$

$$\langle v, kapp(v_0\dots, e_0\ e_1\dots, k) \rangle \mapsto \langle e_0, kapp(v_0\dots v, e_1\dots, k) \rangle$$

$$\langle v_n, kapp(p\ v_0\dots, (), k) \rangle \mapsto \langle \delta(g, v_0\dots v_n), k \rangle$$

$$\delta(\text{SUB}, (v_1\ v_0)) \approx v_0 - v_1$$

(3 4 5)

S-1  $J_1 \rightarrow J_2$

$e := v \mid (e \ e \dots) \mid (\text{if } e \text{ ee}) \mid x$

$x :=$  variable names

$v := b \mid f$

← new

$b :=$  number | bool | prim

$f :=$  function names

$p := (\text{program } (d \dots) \ e)$

$d := (\text{define } (f \ x \dots) \ e)$

(program (define (add1 x) (+ 1 x))  
(add1 5))

$E := \text{hole} \mid (\text{if } E \text{ ee}) \mid (v \dots E \ e \dots)$

$J_1:$  step :  $e \rightarrow e$

$J_2:$  step :  $\Delta \times e \xrightarrow{\quad} e$   
 $(f \mapsto d)$

$E[(f \ v \ \dots)] \mapsto E[e[x_0 \leftarrow v] \ \dots [x_n \leftarrow v_n]]$

where  $\Delta(f) = (\text{define } (f \ x_0 \dots x_n) \ e)$

S-2/  $e[x \leftarrow v]$  means look inside of  $e$ ,

find all the  $x$ 's and replace

$$x[x \leftarrow v] = v \quad \text{with } v$$

$$y[x \leftarrow v] = y \quad (y \notin x)$$

$$u[x \leftarrow v] = u \quad (u \in v \text{ is set})$$

$$(if \ e_1 \ e_2 \ e_3)[x \leftarrow v] = (if \ e_1[x \leftarrow v] \ e_2[x \leftarrow v] \\ e_3[x \leftarrow v])$$

$$(e_0 \dots e_n)[x \leftarrow v] = (e_0[x \leftarrow v] \dots e_n[x \leftarrow v])$$

$$\begin{aligned} f(x) &= \overbrace{7x} + \overbrace{2x^2} + 1 \\ f(5) &= 7 \cdot 5 + 2 \cdot 5^2 + 1 \end{aligned}$$

$$\begin{aligned} & (\text{define } (f \ x \ y) \ (+ \ (* \ x \ 2) \ (- \ x \ y))) \Big] = e \\ & (\text{define } (g \ x) \ (f \ x \ x)) \\ & (+ \ 5 \ (g \ 10) \ (f \ 9 \ (g \ 1))) \quad = e \\ & (\ " \ (f \ 10 \ 10) \ " \ ) \\ & (\ " \ (+ \ 10 \cdot 2 \ -10) \ " \ ) \\ & (+ \ 5 \ 10 \ " \ ) \\ & (+ \ 5 \ 10 \ (f \ 9 \ 1)) \\ & (\ " \ (+ \ 9 \cdot 2 \ -1) \ ) \\ & (+ \ 5 \ 10 \ 17) \end{aligned}$$

S-3)  $C_{K_0} \Rightarrow C_{K_1}$  st =  $\langle \Delta, e, k \rangle$

(1+N)

$\langle \Delta, v_n, kapp((f\ v_0\dots), (), k) \rangle$

$\mapsto \langle \Delta, e[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], k \rangle$

where  $\Delta(f) = (\text{define } (f\ x_0\dots x_n)\ e)$

$\langle \Delta, (\text{if } e_c\ e_t\ e_f), k \rangle \mapsto \langle \Delta, e_c, k \text{ if } (e_t, e_f, k) \rangle$

(if  $\begin{cases} (x > 10 \text{ mil}) \\ \text{true} \end{cases}$  (error))

(define (f x) (f x)) )

(f 0)

$\mapsto (f \overset{1}{0}) \mapsto (f \overset{2}{0})$

"Jay"

proper function call implementation

"Guido"

tail-call optimization

G-1/ CK<sub>i</sub> is linear-time, so it's not fast !!  
and unrealistic because Syntax is available  
at run-time

goal: machine with constant function calls

$\mapsto \langle \Delta, e [x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], h \rangle$

"x"  $f(x) = 2x^2 + 5x + 7$

$\langle f(5) \rangle \mapsto \langle 2x^2 + 5x + 7, [x \leftarrow 5] \rangle$

$\langle 2 \cdot 5^2 + 5 \cdot 5 + 7, [x \leftarrow 5] \rangle$

$\langle 50 + 5 \cdot 5 + 7, [x \leftarrow 5] \rangle$

$\langle 82, [x \leftarrow 5] \rangle = 82$

## 6-2) CK<sub>i</sub> $\Rightarrow$ CEK<sub>0</sub>

$\text{st} = \langle \Delta, e, \text{env}, k \rangle$

$\text{env} = \text{mt} \quad | \quad \text{env}[x \leftarrow v]$

Jay's env vars c = make-env-empty | make-env-consts( $x, v$ ),  
 $\text{env}$ )

inject<sup>A</sup>e =  $\langle \Delta, e, \text{mt}, k \text{ net} \rangle$

extract  $\langle \Delta, e, \text{env}, k \rangle = E(k)[e[\text{env replace}]]$

done?  $\langle \Delta, v, \text{env}, k \text{ net} \rangle$

!!!  $\downarrow$  WRONG  $\downarrow$  !!! (look at head 2 args)

$\langle \Delta, \cancel{x}, \text{mt}, k \rangle \mapsto \text{error}$   $\text{env}[x \leftarrow v]$

$\langle \Delta, x, \text{env}[x \leftarrow v], k \rangle \mapsto \langle \Delta, v, \overset{\text{on}}{\text{mt}}, k \rangle$

$\langle \Delta, x, \text{env}[y \leftarrow v], k \rangle \mapsto \langle \Delta, x, \text{env}, k \rangle$

$\langle \Delta, (\text{if } e_t \text{ } e_s \text{ } e_f), \text{env}, k \rangle \mapsto$

$\langle \Delta, \text{ec}, \text{env}, \text{kif}(e_t, e_f, k) \rangle$

$\langle \Delta, \text{true}, \text{env}, \text{kif}(e_t, e_f, k) \rangle \mapsto \langle \Delta, e_t, \text{env}, k \rangle$

$\langle \Delta, \text{false}, \text{env}, \text{kif}(e_t, e_f, k) \rangle \mapsto \langle \Delta, e_f, \text{env}, k \rangle$

$\langle \Delta, (e_0 \text{ } e_1 \dots), \text{env}, k \rangle \mapsto \langle \Delta, e_0, \text{env}, \text{kapp}((\ ), (e_1 \dots)), k \rangle$

$\langle \Delta, v_n, \text{env}, \text{kapp}((v_0 \dots), (e_0 \dots)), k \rangle$

$\mapsto \langle \Delta, e_0, \text{env}, \text{kapp}((v_0 \dots v_n), (e_1 \dots)), k \rangle$

$\langle \Delta, v_n, \text{env}, \text{kapp}((P v_0 \dots), (\ ), k) \rangle \mapsto \langle \Delta, \delta(P, (v_0 \dots v_n)), \text{env}, k \rangle$

$\langle \Delta, v_n, \text{env}, \text{kapp}((F v_0 \dots), (\ ), k) \rangle \mapsto \text{let } (\text{define } f \text{ } x_0 \dots) \text{ } e = \Delta \text{ in }$

$\langle \Delta, e, \cancel{\text{env}}[x_0 \mapsto v_0] \dots, k \rangle$

6-3) (define (f x) (+ x y))  
(define (g y) (f 5))  
(g 10)

(g 10) → (f 5) → (+ 5 y)

$\langle (g 10), \emptyset \rangle \mapsto \langle (f 5), \text{mt}[y \leftarrow 10] \rangle$   
 $\mapsto \langle (+ x y), \text{mt}[y \leftarrow 10][x \leftarrow 5] \rangle$   
 $\mapsto \langle (+ 5 y), \text{ " } \rangle$   
 $\mapsto \langle (+ 5 10), \text{ " } \rangle$   
 $\mapsto \langle 15, \text{ " } \rangle$

WRONG:

$\langle \Delta, (\text{if } e_t \text{ et } e_f), \text{env}; k \rangle$   
 $\mapsto \langle \Delta, e_t, \text{env}; \text{kif}(e_t, e_f, k) \rangle$   
 ~~$\langle \Delta, \text{true}, \text{env}; \text{kif}(e_t, e_f, k) \rangle$~~   
 $\mapsto \langle \Delta, \text{et}, \text{env}; k \rangle$

(define (g y) true)  
(~~def~~ <sup>func</sup> (if (g 10) y 6))  
(f 6)

Dynamic

Scope

6-4)  $k := \text{kret} \mid \text{kif}(e, e, \text{env}, k)$

$\mid \text{kapp}((\lambda \dots), (e \dots), \text{env}, k)$

RIGHT

$\langle \Delta, x, \text{env}, k \rangle \mapsto \langle \Delta, \text{env}(x), \text{mt}, k \rangle$

$\langle \Delta, \text{if } e_t \text{ et } e_f, \text{env}, k \rangle \mapsto \langle \Delta, e_t, \text{env}, \text{kif}(e_t, e_f, \text{env}, k) \rangle$

$\langle \Delta, \text{true}, \overset{\text{NEW}}{\text{env}}, \text{kif}(e_t, e_f, \text{env}, k) \rangle \mapsto \langle \Delta, e_t, \overset{\text{OLD}}{\text{env}}, k \rangle$

$\langle \Delta, \text{false}, \_, \text{kif}(e_t, e_f, \text{env}, k) \rangle \mapsto \langle \Delta, e_f, \text{env}, k \rangle$

$\langle \Delta, (e_0 \ e_1 \ \dots), \text{env}, k \rangle \mapsto \langle \Delta, e_0, \text{env}, \text{kapp}(\lambda, (e_1 \dots), \text{env}, k) \rangle$

$\langle \Delta, v_n, \_, \text{kapp}((v_0 \dots), (e_0 \ e_1 \ \dots), \text{env}, k) \rangle$

$\mapsto \langle \Delta, e_0, \text{env}, \text{kapp}((v_0 \dots v_n), (e_1 \dots), \text{env}, k) \rangle$

$\langle \Delta, v_n, \_, \text{kapp}((p \ v_0 \ \dots), (), \_, k) \rangle$

$\mapsto \langle \Delta, \delta(p, (v_0 \dots v_n)), \text{mt}, k \rangle$

$\langle \Delta, v_n, \_, \text{kapp}((f \ v_0 \ \dots), (), \_, k) \rangle$

$\mapsto \langle \Delta, e, \text{mt} [x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], k \rangle$

where  $\Delta(f) = (\text{define } (f \ x_0 \dots x_n) \ e)$

6-5)  $\mathcal{J}_2 \rightarrow \mathcal{J}_3$

$e := v \mid (e e \dots) \mid (\text{if } e e_1) \mid x$

$v := b \mid (\lambda (x \dots) e)$

$x := \text{variable names}$

$b := \text{number} \mid \text{bool} \mid \text{prim}$

$E[(\lambda(x_0 \dots x_n) e) v_0 \dots v_n]$

$\mapsto E[e[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n]]$

capture-avoiding

$((\lambda(x) x) s) \mapsto s$

$((\lambda(x) (\lambda(y) x)) s) 6) \mapsto$

$((\lambda(y) s) 6) \mapsto s$

$((\lambda(f) (\lambda(y) f s)) (\lambda(x) y)) 6) 7) \mapsto$

$((\lambda(y) (\lambda(x) y)) 6) 7) \mapsto$

$((\lambda(x) 6) 7) \mapsto 6$

6-6/

$\langle v_n, \text{Dnv}, \text{kapp}((\lambda(x_0 \dots x_n) e) v_0 \dots), (), \text{env} \rangle$   
 $\mapsto \langle e, \text{mt}[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], k \rangle$

$\langle ((\lambda x. y. x) s) 6, \emptyset, k \rangle \quad \text{kapp}(((), (6), \emptyset, k))$

$\langle \lambda x. y. x, \emptyset, \text{kapp}(((), (s)), \emptyset, k) \rangle$

$\langle s, \emptyset, \text{kapp}((\lambda x. y. x), (), \emptyset, \text{kapp}(((), (6), \emptyset, k))) \rangle$

$\langle y, x, \emptyset[x \leftarrow s], \text{kapp}(((), (6), \emptyset, k)) \rangle$

$\langle 6, \emptyset, \text{kapp}((y, x), (), \emptyset, k) \rangle$

$\langle x, \emptyset[y \leftarrow 6], k \rangle$

old

$\langle x, \emptyset[x \leftarrow 6][y \leftarrow 6], k \rangle$

$\langle s, \text{mt}, k \rangle$

$\underbrace{\langle ((\lambda x. y. x) s) 6 \rangle}_{\mapsto 5} \mapsto (y, 6) \quad 6 \mapsto 5$

CEK

old  $v := \text{num} \mid \text{bool} \mid \text{prim} \mid (\lambda(x\dots)e)$

new  $v := \text{num} \mid \text{bool} \mid \text{prim} \mid \text{closure}(\lambda(x\dots)e, \text{env})$

$\langle (\lambda(x\dots)e), \text{env}, k \rangle \mapsto \langle \text{closure}((\lambda(x\dots)e), \text{env}), \emptyset, k \rangle$

$\langle v_n, \_, \text{kapp}((\text{closure}((\lambda(x_0 \dots x_n) e), \text{env}), v_0 \dots), (), \text{env}) \rangle$

$\mapsto \langle e, \text{env}[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n], k \rangle$

6-7)

desugar (let ([x<sub>0</sub> e<sub>0</sub>] ... [x<sub>n</sub> e<sub>n</sub>]) be)

= ((λ (x<sub>0</sub> ... x<sub>n</sub>) be) e<sub>0</sub> ... e<sub>n</sub>)

(let ([x s] [y 6]))

(+ x y))

=

((λ (x y) (+ x y))  
s 6)

(let ([x ← s]))

(+ x ↙ ↘ )

(let ↘ ([x (+ 1 x)]) (+ x x))

(+ x y)))

∅[x ← s][x ← 6]

= (+ s

(let ([x ← (+ 1 s)]) (+ x x))

(+ s y))

= (+ s (+ 6 6) (+ s y))

= (+ s 12 q)

= 26

6-8) desugar (let\* () be) =  $\lambda e$   
 desugar (let\* ([ $x_0 e_0$ ] [ $x_1 e_1$ ] ...) be)  
 = (let ([ $x_0 e_0$ ])  
 (let\* ([ $x_1 e_1$ ] ...) be))

(let\* ( $\underbrace{[x \leftarrow s]}$   
 $\underbrace{[y \leftarrow [ (+ x 1) ]}$   
 $\underbrace{[z \leftarrow [ (+ x y) ])}$   
 $(+ z \underbrace{z})$ )

normal let:  $((\lambda (x y z) (+ z z))$   
 $\quad \quad \quad s (+ x 1) (+ x y))$   
 $\text{let* } : \lambda^{(x)} ((\lambda(y) ((\lambda(z) (+ z z) (+ x y)))$   
 $\quad \quad \quad (+ x 1))) s)$

7-1 / ((let ([x←5])  

$$(\lambda(y)(+x^y)))$$
  

$$\overbrace{6)} \Rightarrow (+ 5 6)$$

$$\Delta : e = x \quad | \quad \lambda x. e \quad | \quad e \quad e$$

*id :  $\lambda x. x$*

$$(\lambda x. \lambda y. x) (\lambda z. z) (\lambda g. g) \Rightarrow (\lambda z. z)$$

## Alonzo Church's Lambda Calculus

Church encoding == Object-oriented Prog.

## Representas interface

## interface vs Appresentazione

`bool : Opt1 → Opt2 → Some Option 1 or 2`

true :=  $\lambda x. \lambda y. x$

false := Ax. Ay. y

$i_f := \lambda b. \lambda t. \lambda f. b + f$

if tree  $M \circ N \Rightarrow (Ab + f, b + c)$  tree  $M \circ N$

$$\Rightarrow \text{true} \wedge N \Rightarrow (\lambda xy.x) \wedge N \Rightarrow M$$

## 7-2/ Church-encoded numbers

numbers are iteration

num : ThingToDo  $\Rightarrow$  SomethingToDoItTo  
 $\rightarrow$  Result of doing it N times

$$\text{zero} := \lambda f. \lambda z. z$$

$$\text{one} := \lambda f. \lambda z. f z$$

$$\text{two} := \lambda f. \lambda z. f(f z)$$

$$\text{add1} := \lambda n. \lambda f. \lambda z. f(n f z)$$

$$\begin{aligned}\text{add1 zero} &\Rightarrow \lambda f. \lambda z. f(\text{zero } f z) \\ &\Rightarrow \lambda f. \lambda z. f z = \text{one}\end{aligned}$$

$$\text{plus} := \lambda n. \lambda m. \lambda f. \lambda z. m f (n f z)$$

$$\begin{aligned}\text{plus one one} &\Rightarrow \lambda f. \lambda z. \text{one } f (\text{one } f z) \\ &\Rightarrow \lambda f. \lambda z. \text{one } f (f z) \\ &\Rightarrow \lambda f. \lambda z. f(f z) = \text{two}\end{aligned}$$

$$\text{mult} := \lambda n. \lambda m. \lambda f. \lambda z. n(m f) z$$

$$\text{mult two two} \Rightarrow \lambda f. \lambda z. \text{two } (\text{two } f) z$$

$$\lambda f. \lambda z. (\text{two } f) (\text{two } f) z$$

$$\lambda f. \lambda z. f(f(f(f z)))) =$$

four

7-3/ zero? =  $\lambda n. \; n \; (\lambda x. \text{false}) \; \text{true}$

$$\begin{aligned} \text{zero? zero} &\Rightarrow \text{zero } (\lambda x. \text{F}) \; \text{T} \\ &\Rightarrow \text{T} \end{aligned}$$

$$\begin{aligned} \text{zero? one} &\Rightarrow \text{one } (\lambda x. \text{F}) \; \text{T} \\ &\Rightarrow (\lambda x. \text{F}) \; \text{T} \\ &\Rightarrow \text{F} \end{aligned}$$

$$\begin{aligned} \text{fst } (\text{pair } M \; N) &\Rightarrow M \\ \text{snd } (\text{pair } M \; N) &\Rightarrow N \end{aligned}$$

$$\text{pair} := \lambda x. \lambda y. \lambda \text{sel}. \; \text{sel} \times y$$

$$\text{fst} := \lambda p. \; p \; \text{true}$$

$$\text{snd} := \lambda p. \; p \; \text{false}$$

$$\begin{aligned} \text{fst } (\text{pair } M \; N) &\Rightarrow (\text{pair } M \; N) \; \text{true} \\ &\Rightarrow (\text{true } M \; N) \Rightarrow M \end{aligned}$$

---

$$\text{select } (\text{int } M) \; f \; g \Rightarrow f \; M$$

$$\text{select } (\text{inn } N) \; f \; g \Rightarrow g \; N$$

$$\text{int} := \lambda v. \lambda f. \lambda g. \; f \; v$$

$$\text{inn} := \lambda v. \lambda R. \lambda g. \; g \; v$$

$$\text{select!} := \lambda o. \lambda f. \lambda g. \; o \; f \; g$$

copy :=  $\lambda n. n$  add1 zero

74/ sub1 :=  $\lambda n. \text{fst}(\text{F } (\text{pair zero zero}))$

F :=  $\lambda p. \text{pair } (\text{snd } p) \ ( \text{add1 } (\text{snd } p))$

sub :=  $\lambda n. \text{m. m sub1 n}$

fac :=  $\lambda n.$

$\lambda$  if (zero? n)

defn

one



(mult n (fac (sub1 n)))

mkfac :=  $\lambda \text{fac. }$

$\lambda n. \text{if } (\text{zero? } n) \ \text{one}$

$(\text{mult } n (\text{fac } (\text{sub1 } n)))$

fac :=  $\lambda$  mkfac

$\lambda$  mkfac  $\Rightarrow$  fac

$\lambda ( \lambda \text{fac. } M ) \Rightarrow M [ \text{fac} \leftarrow (\lambda \text{mkfac}) ]$

$\lambda ( \lambda x. M ) \Rightarrow M ( \lambda ( \lambda x. M ))$

$\leftarrow$  fixed-point combinator (eager) aka Y

Z :=  $\lambda f. ( \lambda x. (f (\lambda v. (x \ x \ v))))$   
 $(\lambda x. (f (\lambda v. (x \ x \ v))))$

$Z \ f = f (Z \ f)$

$$\boxed{Z-5} / \Omega_1 = w \quad w$$

$$w := \lambda x. x \quad x$$

$$\text{eg } \Omega \Rightarrow w \quad w \Rightarrow$$

$$(\lambda x. (x \ x)) \quad (\lambda x. (x \ x))$$

$$\Rightarrow (\lambda x. (x \ x))' \quad (\lambda x. (x \ x))$$

$$\Rightarrow \quad w \quad w$$

$$\Rightarrow \quad \Omega$$

The Lambda Calculus  $\subseteq S_2$

A  $S_2$  to convert Church to Normal :=

$$\text{Church2Normal} := \lambda n. n \ (\lambda x. (+\ 1\ x)) \ 0$$

$$\text{Church2Normal} \quad (\text{fac} \ (\text{succ} \ (\text{plus} \ \text{two} \ \text{two})))$$

$$\Rightarrow 120$$

8-1) J<sub>2</sub>: (define (even? x)  
 (if (zero? x) true  
~~(odd? (sub1 x)))~~  
 (define (odd? x)  
 (if (zero? x) false  
 (even? (sub1 x))))  
 (even? 10))

$$\Delta = \{ \text{even?}, \text{odd?} \}$$

✓ odd? is undefined

J<sub>3</sub>: (let\* ([even? (lambda (x) ... odd? ...)])  
 [odd? (lambda (x) ... even? ...)])  
 (even? 10))

J<sub>3</sub>  $\Rightarrow$  J<sub>4</sub> = v := ... | (lambda (x ...) e)

recursive name of fun

(let ([fac (lambda (n) (if (= n 0) 1  
 (\* n (ifac (- n 1)))))])  
 (fac 5))

(desugar (lambda (x ...) e))  $\Rightarrow$  (desugar (lambda (x ...) e))

8-2/ GLD:

$$E[\lambda v_0 \dots v_n] = E[e[x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n]]$$

$$\text{where } \lambda = (\lambda (x_0 \dots x_n) e)$$

NEW:

$$E[\lambda v_0 \dots v_n] = E[e[f \leftarrow \lambda] [x_0 \leftarrow v_0] \dots [x_n \leftarrow v_n]]$$

$$\text{where } \lambda = (\lambda f (x_0 \dots x_n) e)$$

CEK<sub>1</sub>  $\rightarrow$  CEK<sub>2</sub>

OLD:  $\langle (\lambda (x_0 \dots x_n) e), \text{env}, k \rangle$

$\mapsto \langle \text{clo}(\lambda (x_0 \dots x_n) e), \text{env}, mt, k \rangle$

$\langle \lambda, \text{env}, k \rangle \mapsto \langle \text{clo}(\lambda, \text{env}), mt, k \rangle$

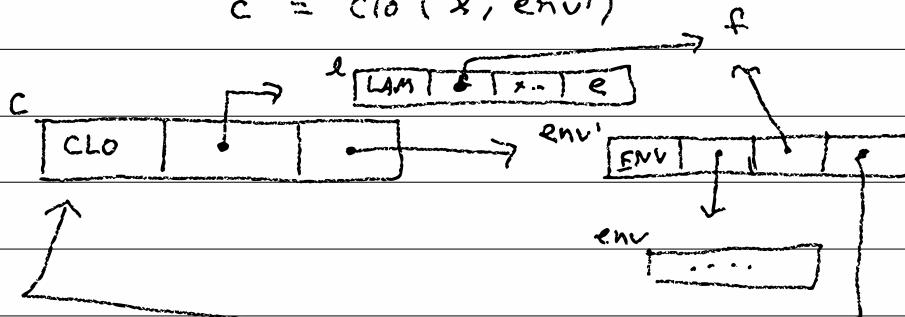
$$\text{where } \lambda = (\lambda (x_0 \dots x_n) e)$$

NEW:  $\langle \lambda, \text{env}, k \rangle \mapsto \langle c, mt, k \rangle$

$$\text{where } \lambda = (\lambda f (x_0 \dots x_n) e)$$

$$\text{env}' = \text{env}[f \leftarrow c]$$

$$c = \text{clo}(\lambda, \text{env}')$$



8-3/  $\text{envp} = \text{make\_env\_ext}(\text{env}, l \mapsto f, \text{NULL});$   
 $\text{clo} = \text{make\_rto}(l, \text{envp});$   
 $\text{envp} \rightarrow \text{val} = \text{clo}; / \backslash \leftarrow \text{installs cycle!}$

as  $\text{nat-unfold} :=$

$(\lambda \text{ rec } (f \in n))$

$(\text{if } (= n 0) z$

$(f n (\text{nat-unfold}^{\text{rec}} f z (-n))))$

$\text{fac } n = \text{nat-unfold } (\lambda (n a) ((\& n a)) 1 n$