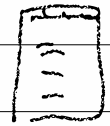
(-1)

Goal: Implement a VM

What does a program mean?

"1+1" means "2"

"wordle" means 

a semantics
- english (human)
- math (universal)
- code

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

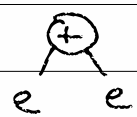
BNF or CFG

$\mathcal{J}_0$: e := v | (+ e e) | (* e e)

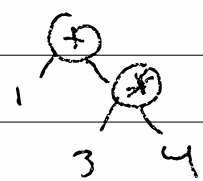
v := number

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

"(+ 1))"

e := v or  or 

v := number



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

```
class   E   {}                  new Mult(
class   Add : E {}                  new Num(3),
class   Num : E {}                  new Num(4))
class   Mult : E {}

Add :: Add(E *l, E *r) {
    this.l = l; this.r = r; }
```

(+ 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 (a big step semantics) : e → v

interp v = v

interp (+ e_1 e_2) = (interp e_1) $+_v$ (interp e_2)

$+_v$ (Num n_1) (Num n_2) = Num ($n_1 + n_2$)

1-3)

E^* ← Num* = V*

```
virtual E::interp() = 0;
// interp v = v
E* Num::interp() { return this; }
// interp (+ e1 e2) = (interp e1) +v (interp e2)
E* Add::interp() {
Num* n1 = this.l.interp();
Num* n2 = this.r.interp();
  return new Num(n1.n + n2.n); }
```

(- e1) == (* -1 e1)

(- e1 e2) = (+ e1 (- e2))

desugar (expander) [compiler] : string tree

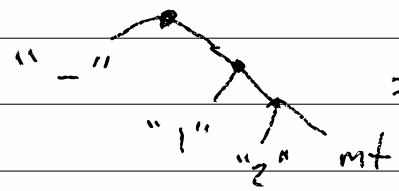
string expr

Sexpr = string or pair of sexpr or empty

S-expr

Sexpr

⟹ T_0



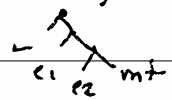
= (- 1 2)

1 - 2

2-1/

Sexpr $\quad\quad\quad\quad$ J_0

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



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

$(+) = 0$

$(+\ e_1\ .\ more) = (+\ (d\ e_1)\ (d\ (+\ .\ more)))$

$(*) = 1$

$(*\ e_1\ .\ more) = (*\ (d\ e_1)\ (d\ (*\ .\ more)))$

J_1 $\quad e ::= v \mid (\text{if } e\ e\ e) \mid (e\ e \ldots)$

$v ::=$ number $\mid$ bool $\mid$ prim

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

interp $v = v$

interp (if e_c e_t e_e) =

$c =$ interp e_c

$e_k =$ <u>if</u> c e_t <u>or</u> e_f

return interp e_k

interp $(e_p\ e_0 \ldots e_n) =$

$p =$ interp e_p (must be a prim)

$v_0 \ldots v_n =$ interp $v_0 \ldots$ interp v_n

ret $\delta(p, v_0 \ldots v_n)$

2-2/

```
v* delta (prim p, list v* vs) =
  if (p == ADD)
      return new Num(vs[0].n + vs[1].n)
  if (p == LT)
      return new Bool(vs[0].n < 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 wrt proving
- inefficient / unhelpful for implementation

small step : $e \to e$

step (if true e_t e_f) = e_t

step (if false e_t e_f) = e_f

step (p v_0 ... v_n) = $\delta(p, v_0 \ldots v_n)$

if step e_c = e_c' then sstep (if e_c e_t e_f)
= (if e_c' e_t e_f)

if step e_i = e_i' then step (e_0 ... e_i e_{i+1} ... e_n)
= (e_0 ... e_i' e_{i+1} ... e_n)

2-3)

$$(1+1) + \overbrace{(2+3)}^{\text{step}} = 2 + \overbrace{(2+3)}^{\text{step}} = \overbrace{2+5}^{\text{step}} = 7$$

$$= (1+1) + 5$$

A context, C ::= hole

| if C e e

| if e C e

| if e e C

| (e ... C e ...)

plug : C × e ⇒ e

plug hole e = e

plug (if C e_1 e_2) e_p = (if (plug C e_p) e_1 e_2)

plug (e_b ... C e_a ...) e_p = (e_b ... (plug C e_p) e_a ...)

plug (+ (2+3)) (1+1) = (1+1) + (2+3)

C 2 = 2 + (2+3)

Parse : e ⇒ C × e

2-4/

$e \Rightarrow e$

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

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

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

find_redex : $e \Rightarrow C \times e$

↑ redex

reducible expression

When do two programs mean the same thing?

"1 + 2" "2+1"

"4 billion + 1 + 2" "2 + 1 + 4 billion"

$C = \blacksquare$

"quicksort" "mergesort" "heapsort"

"insertion sort"

$\forall l.$ "mergesort l" = "hs l"

$C = (\blacksquare\ L)$

$\forall C.\ \overset{\text{interp}}{C[x]} = \overset{\text{interp}}{C[y]}$ — observational equivalence

time e = e's ans x secs