

Reflective Programs - programs in a language that can operate on themselves

↳ Programs then are both functions and data structures

Turing showed how to represent a program on a tape

↳ But needed a universal machine to process it.

Church - requires external machinery to interpret the functions as data structures

Syntax Trees  $\rightarrow$  Source Programs  $\rightarrow$  Executable Functions  
(Data Structures)

Application Developers cannot do their own static analysis

↳ requires building new programming languages

Node Operator:  $\Delta$

↳ left associative applications

$$\Delta y z = y$$

$$\Delta(\Delta x) y z = (\Delta x)(y z)$$

$$\Delta(\Delta w x) y z = z w x$$

Program Analysis is meta-theoretic

Backus-Naur Form:

$n := \text{zero} \mid \text{successor } n$

BNF for Roman Numerals:

$r, s := I \mid V \mid X \mid L \mid C \mid D \mid M \mid (r \ s)$

Arithmetic expression:

$m, n := \text{zero} \mid \text{successor } m \mid (\text{arith})$

↳ equality relation is an

(1) Equivalence (reflexive, symmetric, transitive)

(2) Congruent

Theorem 1:

If  $m, n$  are unary numerals then  $\exists p$  s.t.  
 $m \equiv_p n$

Theorem 2: Every arithmetic expression is equal to a numeral

Theorem 3: For all numerals  $m, n$  we have that

$$m + \text{succ}(n) = \text{succ}(m+n)$$

Pf

For  $m=0$ ,  $0 + \text{succ}(n) = \text{succ}(n) = \text{succ}(0+n)$

If  $m = \text{succ } m_1$ , then

$$\begin{aligned} \text{succ } m_1 + \text{succ } n &= m_1 + \text{succ}(\text{succ } n) \\ &= \text{succ}(m_1 + \text{succ } n) \end{aligned}$$

Theorem 4: Zero is unit for addition of any unary numerals:

$$0+n = n = n+0$$

If  $n = \text{succ } m$ , then

$$\begin{aligned}0+n &= 0+\text{succ } m_1 = (\text{succ } 0)+m_1 \\&= \text{succ}(m_1+0) = \text{succ } m_1 \\&= n = n+0\end{aligned}$$

Theorem 5: Addition is commutative

$$e_1 + e_2 = e_2 + e_1$$

Theorem 6: Addition is associative

$$(e_1 + e_2) + e_3 = e_1 + (e_2 + e_3)$$

Translation from Roman Numerals to Unary

$$[I] = \text{succ } 0$$

$$[V] = [I] + [I] + \dots + [I]$$

$$[X] = [V] + [V]$$

:

:

$$[r+s] = [r] + [s]$$

Theorem 7.  $r=s \Leftrightarrow [r] = [s]$

pf

Let  $r_1 = s_1, r_2 = s_2$ , then

$$[r, r_2] = [r_1] + [r_2] = [s_1] + [s_2] = [s, s_2]$$

## Chapter 2. Tree Calculus:

Without syntax trees, programs are just unlabelled (natural) trees

Tree calculus

↳ 1 operator, 3 eval rules

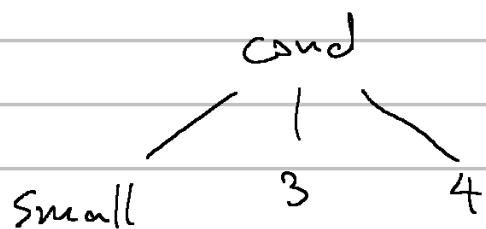
Normal Program

[i, f, s, m, a, /, /, then, n, 3, else, s, e, 4]

↓ tokenized

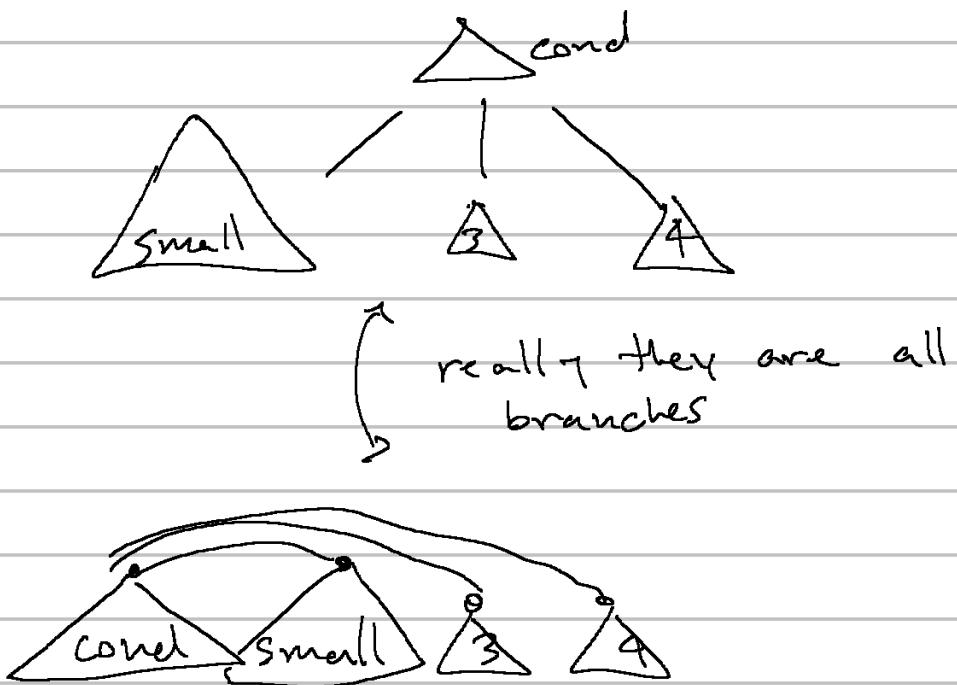
[if small, then, 3, else, 4]

↓ syntax tree



Reflective programming lets you do this analysis  
after the program becomes an executable  
↳ syntax tree from an executable

\* Instead of converting trees to executables,  
the trees themselves are executables



Grafting a new branch  $N$  to an existing tree  $M$  is represented by the application  $MN$   
(left-associative)

Natural trees follow the BNF:  $M, N ::= \Delta \mid MN$

Since tree calculus is reflective, programs should also qualify as inputs

Values / Programs are a single subclass of trees

Leaf / Kernel := Tree w/o branches

Branch := Tree w/ one branch

Fork := " w/ 2 branches

⇒ Values are chains of the form

$\Delta(\Delta(\dots(\Delta\Delta)))$  correspond to the natural nms.

$M^k(N) = MCMC\dots(MN)\dots$

Chains have  
no obvious internal

↪ Identify  $k \in \mathbb{N}$  with  $\Delta^k \Delta$  structure

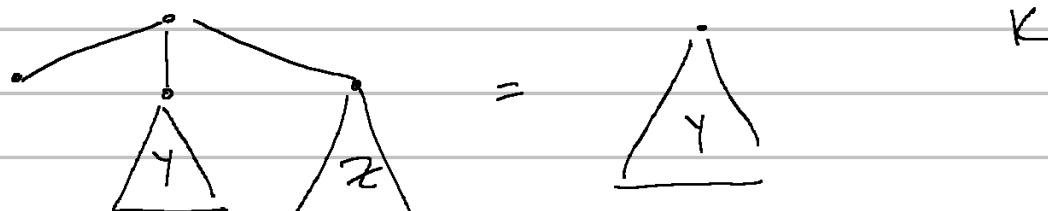
- Tree Calculus uses only Binary Trees!
- ↳ A tree  $\Delta MNP$  has to be evaluated
  - ↳ If  $\Delta MN$  is a program then  $P$  is the input
  - ↳  $\Delta$  is a ternary operator

If eval doesn't require checking internal properties, then we get

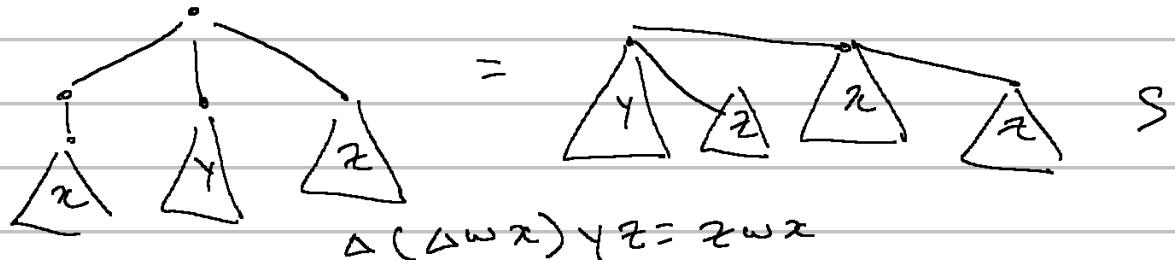
$$\Delta xyz = t \text{ tree}$$

↳ But then this is just combinatorial logic!

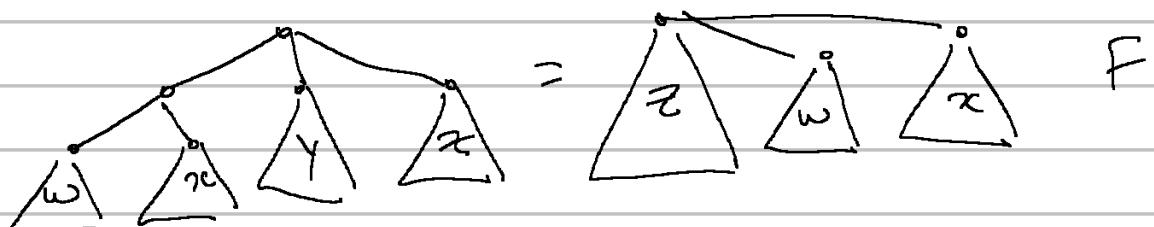
$$\Delta\Delta yz = y$$

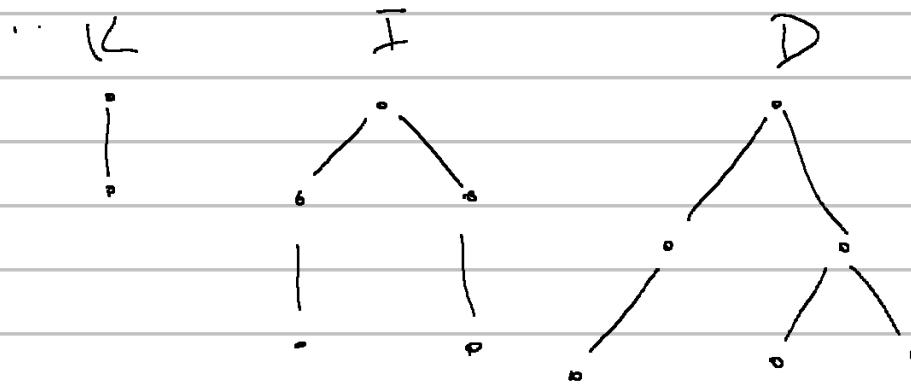


$$\Delta(\Delta x)yz = (yz)(xz)$$



$$\Delta(\Delta w x)yz = zwz$$





$$L = \Delta \Delta$$

$$I = \Delta (\Delta \Delta) (\Delta \Delta)$$

$$D = \Delta (\Delta \Delta) (\Delta \Delta \Delta)$$

So then

$$Lyz = \Delta \Delta yz = y$$

$$Ix = \Delta (\Delta \Delta) (\Delta \Delta) x = \Delta \Delta x (\Delta x) = x$$

$$\begin{aligned} Dx yz &= \Delta (\Delta \Delta) (\Delta \Delta \Delta) xyz \\ &= \Delta \Delta \Delta x (\Delta x) yz \quad \text{rule 2} \quad \Delta (\Delta x) yz \\ &= \Delta x (\Delta x) yz \quad \Delta \Delta yz = y \quad = (yz)(xz) \\ &= yz xz \end{aligned}$$

$$Sxyz = xz(yz) = D_{yxz}$$

$$Sxy = D_{yx}$$

$$= D_y (L_{xy})$$

$$= D(L_{xy}) D_y$$

Programs in tree calculus are given by the BNF

$$p_1, p_2 := \Delta \mid \Delta p_1 \mid \Delta p_2$$

Propositional Logic

$$\underline{\text{true}} \quad xy = x$$

$$\underline{\text{false}} \quad xy = y$$



$$\text{Let } \underline{\text{true}} = K, \underline{\text{false}} = I \quad K = \Delta \Delta (\Delta (\Delta \Delta) (\Delta \Delta))$$

$$\text{Since } KIxy = Iy = y$$

$$d\{x\} = \Delta(\Delta x)$$

Pair:  $\Delta$

$$\text{first}\{p\} = \Delta p \Delta^1 K$$

$$\text{second}\{p\} = \Delta p \Delta^1 (K)$$

So then

$$\text{And} = d\{K(K(I))\}$$

$$\Rightarrow \text{first}\{\text{Pair } xy\} = \Delta(\Delta xy) \Delta^1 K$$

$$\text{OR} = d\{KK\} I$$

$$= Kxy = x$$

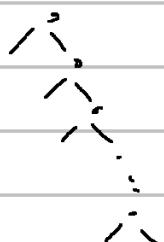
$$\Rightarrow = d\{KK\}$$

$$\Rightarrow \text{second}\{\text{Pair } xy\} = \Delta(\Delta xy) \Delta^1 K \\ = KI(xy) = y$$

$$\text{NOT} = d\{KK\} (d\{K(KI)\} I)$$

$$\Leftrightarrow = \Delta(\Delta I \text{ NOT}) \Delta$$

Natural Numbers:  $K^n \Delta$



$$\begin{aligned}
 \text{isZero} &= d\{k^4 I\} (d\{kk\} \Delta) \\
 &= d\{k^4 I\} (d\{kk\} \Delta) n \\
 &= \Delta n k (k^3 I) \\
 &= \Delta n k (k^3 I)
 \end{aligned}$$

$$\begin{aligned}
 \text{So } \text{isZero } \Delta &= \Delta n k (k^3 I) \\
 &= k = \text{true}
 \end{aligned}$$

$$\begin{aligned}
 \text{isZero}(kn) &= \Delta(kn) k (k^3 I) \\
 &\Delta(\Delta n) k (k^3 I) \\
 &k^3 I \Delta n \\
 &k I = \underline{\text{false}}
 \end{aligned}$$

$$\begin{aligned}
 \text{Successor } n^\Delta &= kn \Delta = n \\
 \text{Predecessor } n &= \Delta n \Delta (k)
 \end{aligned}$$

Fixpoint function: Any function  $f$  for which the program  $\gamma_2\{f\}$  is a fixpoint satisfies

$$\gamma_2 f_n = f_n(\gamma_2 f)$$

Branch first Evaluation:

↳ Assume  $t$  and  $u$  above are programs and produce result from their application