

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 → Source Programs → Executable Functions  
(Data Structures)

Application Developers cannot do their own static analysis

↳ requires building new programming languages

Node Operator:  $\Delta$

↳ left associative applications

$$\Delta \Delta y z = y$$

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

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

Program Analysis is meta-theoretic

Backus-Naur Form:

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

BNF for Roman Numerals:

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

Arithmetic Expression:

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

↳ equality relation is an

(1) Equivalence (reflexive, symmetric, transitive)

(2) Congruent

Thm 1:

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

Thm 2: Every arithmetic expression is equal to a numeral

Thm 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

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

Thm 4: Zero is unit for addition of any unary numerals:  
 $0 + n = n = n + 0$

If  $n = \text{succ } n_1$ , then

$$\begin{aligned} 0 + n &= 0 + \text{succ } n_1 = (\text{succ } 0) + n_1 \\ &= \text{succ}(n_1 + 0) = \text{succ } n_1 \\ &= n = n + 0 \end{aligned}$$

Thm 5: Addition is commutative  
 $e_1 + e_2 = e_2 + e_1$

Thm 6: Addition is associative  
 $(e_1 + e_2) + e_3 = e_1 + (e_2 + e_3)$

Translation from Roman Numerals to Unary

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

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

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

$\vdots$

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

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

Pf

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

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

## Chapter 2. Tree Calculus:

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

Tree calculus

↳ 1 operator, 3 eval rules

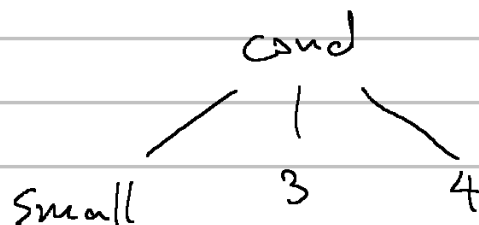
Normal Program

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

↓ tokenized

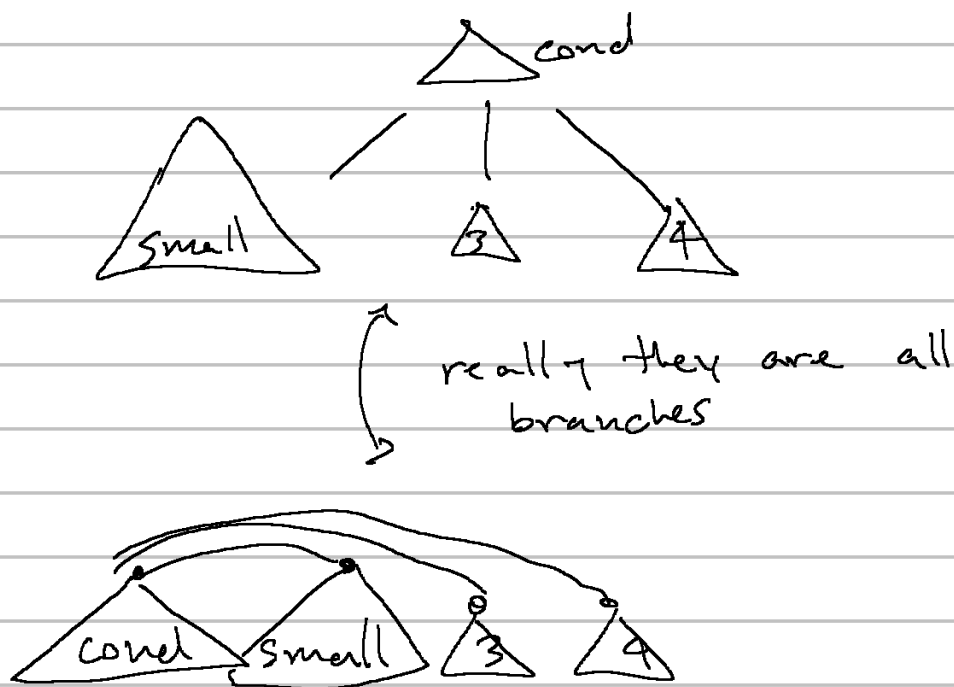
[if, small, then, 3, else, 4]

↓ 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

$\Rightarrow$  Values are chains of the form

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

$M^k(N) = M(M(\dots(MN)\dots))$  Chains have no obvious internal structure

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

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

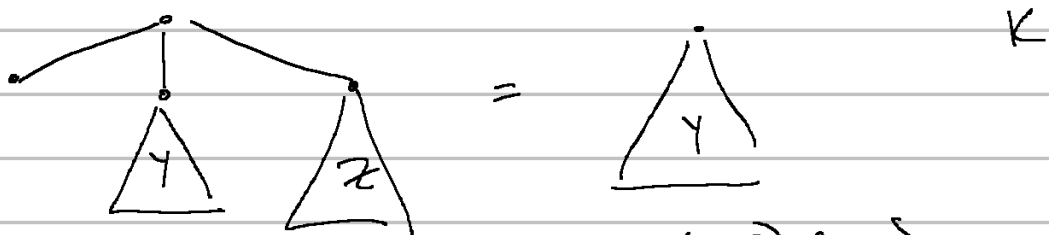
↳  $\Delta$  is a ternary operator

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

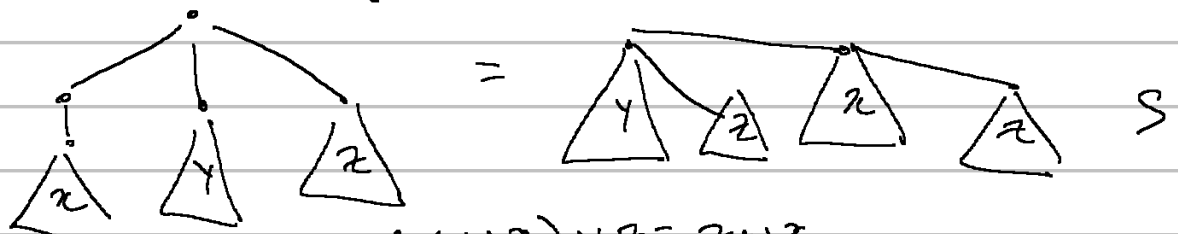
$$\Delta x y z = x \text{ tree}$$

↳ But then this is just combinatorial logic!

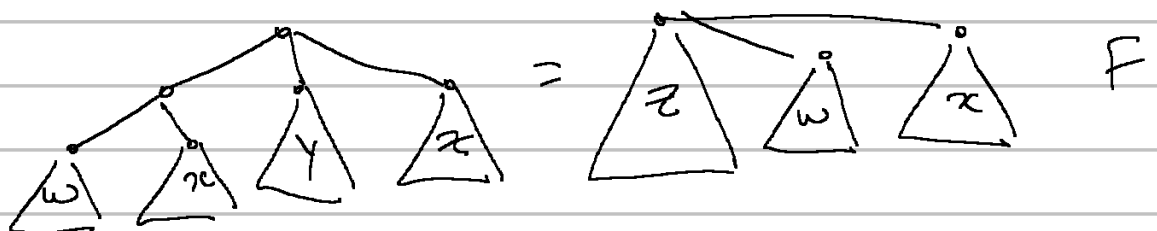
$$\Delta \Delta y z = y$$

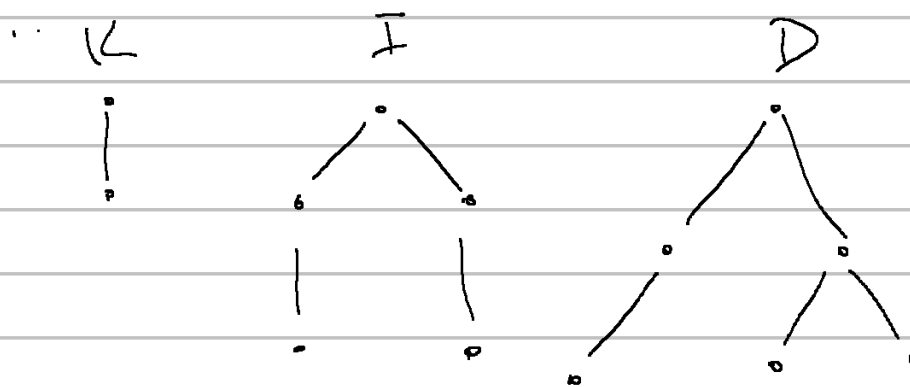


$$\Delta(\Delta x) y z = (yz)(xz)$$



$$\Delta(\Delta wx) y z = zwz$$





$$K = \Delta \Delta$$

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

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

So then

$$Kyz = \Delta\Delta yz = y$$

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

$$\begin{aligned}
 Dxyz &= \Delta(\Delta\Delta)(\Delta\Delta\Delta)xyz \\
 &= \Delta\Delta\Delta x(\Delta x)yz \quad \left. \begin{array}{l} \text{rule 2} \\ \Delta(\Delta x)yz \end{array} \right\} \\
 &= \Delta x(\Delta x)yz \quad \left. \begin{array}{l} \Delta\Delta yz = y \\ \Delta(\Delta x)yz \end{array} \right\} = (yz)(xz) \\
 &= yzxx
 \end{aligned}$$

$$Sxyz = xz(yz) = Dyxz$$

$$\begin{aligned}
 Sxy &= Dyx \\
 &= Dy(Kxy) \\
 &= D(Kx)Dy
 \end{aligned}$$

Programs in tree calculus are given by the BNF

$$p, q := \Delta \mid \Delta p \mid \Delta p q$$

Propositional Logic

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

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



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

Since  $KIx y = Iy = y$

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

Pair:  $\Delta$

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

$$\text{second}\{p\} = \Delta p \Delta (KI)$$

So then

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

$$\Rightarrow \text{first}\{\text{Pair } xy\} = \Delta(\Delta xy) \Delta K = Kxy = x$$

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

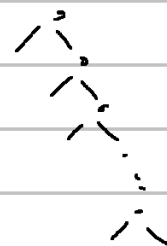
$$\Rightarrow = d\{KK\}$$

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

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

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

Natural Numbers:  $K^n \Delta$



$$\begin{aligned}
\text{isZero} &= d\{K^4 I\} (d\{KK\} \Delta) \\
&= d\{K^4 I\} (d\{KK\} \Delta) u \\
&= \Delta u (KK u) (K^3 I) \\
&= \Delta u K (K^3 I)
\end{aligned}$$

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

$$\begin{aligned}
\text{isZero } (Ku) &= \Delta (Ku) K (K^3 I) \\
&\quad \Delta (\Delta u) K (K^3 I) \\
&\quad K^3 I \Delta u \\
&\quad KI = \text{false}
\end{aligned}$$

$$\text{Successor } n \Delta = K n \Delta = n$$

$$\text{Predecessor } n = \Delta n \Delta (KI)$$

**Fixpoint Function:** Any function  $f$  for which the program  $Y_2\{f\}$  is a fixpoint satisfies

$$Y_2 f x = f x (Y_2 f)$$

Branch-first Evaluation:

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