Seemingly impossible APL programs

Life is always going to be stranger than fiction, because fiction needs to be convincing, and life doesn't.

~ Neil Gaiman

Setting the stage: What is an array?

- Roger Hui and Ken Iverson:
  - ***An array is a function*** *from a set of indices to numbers, characters, … A rank-n array is one whose function f applies to n-tuples of non-negative integers, […]*
- Hence:
  - `⍳∞          → {ω}`
  - `∞⍴1 2 3    → {1 2 3⊃⍨3|ω}`
  - `  1 2 3    → {1 2 3⊃⍨  ω}`
- Nothing new: E. E. McDonnell - "Extending APL to Infinities".

Setting the stage: What is a function?

- Discrete: We can represent function as a relation:
  - `{ω=5: 3 ⋄ ω=1: 8} → ↑(5 3) (1 8)`
- Continuous: If analytic, the function may have a power series.
- Power series: often represented as a vector of coefficients.
- Intuitively speaking: many functions have polynomial approximations. What if we made the polynomial infinitely long?

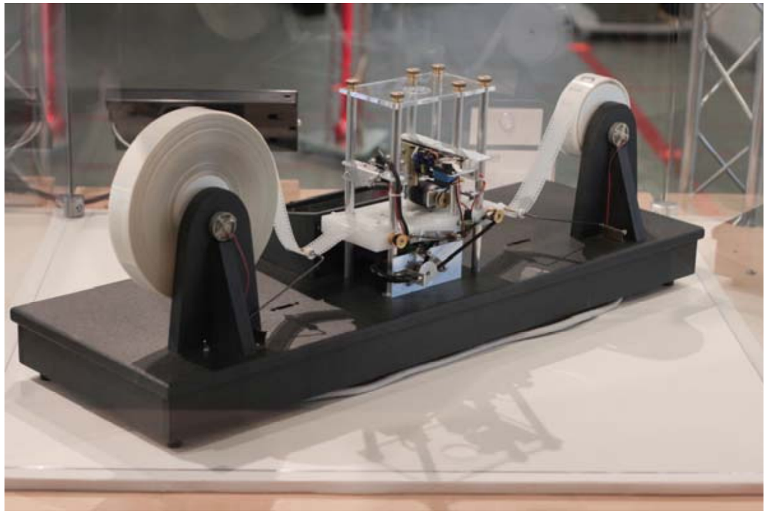
$$\sum_{k=0}^{\infty} a_k x^k$$

~ `x⊥a¨⍳∞`

Roger Hui – *Bring something beautiful*, Vector: Vol. 24, No. 4

Thought experiment.

- Imagine a hypothetical black-box apparatus scanning an infinitely long punched tape containing your message. If the code is considered appropriate, a green lamp turns on; otherwise, the tape is shredded.
- Objective: Craft a message that makes the machine happy.



Mathematical insight.

- The Machine is a function that, given a function that maps from Natural numbers to Bits, determines whether it is suitable or not (also returns a bit): $(N \to B) \to B$.
- Caveat: There are infinitely many possible functions $N \to B$! We can't establish equality between a function over an infinite set, in finite time.

Or can we?

What?

- It is impossible to establish equality between functions **A → B** if **A** is infinite (Turing, Kleene, etc...)
- There are function types over infinite sets that admit decidable equality: For example, $(\boldsymbol{N} \to \boldsymbol{B}) \to \boldsymbol{N}$.
- Topological observations:
  - Finite parts of the output depend on the finite parts of input (Brouwer).
  - Hence: The function is continuous.
- Star of today's show: the Cantor space – $\boldsymbol{N} \to \boldsymbol{B}$.

Ulrich Berger (1990)

```
data Bit = Zero | One
type Cantor = [Bit]
find :: (Cantor -> Bool) -> Cantor
forsome, forevery :: (Cantor -> Bool) -> Bool
find p = if forsome(\a -> p(Zero : a))
       then Zero : find(\a -> p(Zero : a))
       else One : find(\a -> p(One : a))
forsome p = p(find p)
forevery p = not(forsome(\a -> not(p a)))
```

How?

- Rewrite a mutually recursive call:
  - ```
    find p = if p(Zero : find(\a -> p(Zero : a))
        then Zero : find(\a -> p(Zero : a))
        else One : find(\a -> p(One : a))
    ```
- Topological argument:
  - $(\boldsymbol{N} \to \boldsymbol{B}) \to \boldsymbol{N}$ is uniformly continuous (we also assume that it's total, i.e. it terminates).
  - Meaning: There exist such sequences α and ω that there is a minimum m where (m↑α)≡(m↑ω) implies (f α)≡(f ω).
  - m: the *modulus of uniform continuity*.
  - m=0 implies that f and g do not depend on their arguments.
  - Otherwise, the cons predicates have m one smaller.
    ```

APL - Stateless!

```
cantor←{
   C←{(f: (ω.f){ω=0:ωω ◇ αα ω-1}α)}




}
```

APL – Stateless!

```
cantor←{
   C←{(f: (ω.f){ω=0:ωω ⋄ αα ω-1}α)}
   F←{b←αα P 0 ⋄ αα b:b ⋄ αα P 1}




}
```

APL – Stateless!

```
cantor←{
  C←{(f: (⍵.f){⍵=0:⍵⍵ ⋄ ⍺⍺ ⍵-1}⍺)}
  F←{b←⍺⍺ P 0 ⋄ ⍺⍺ b:b ⋄ ⍺⍺ P 1}
  P←{⍵ C (P: P ⋄ C: C ⋄ F: F
     f: (⍺⍺∘(⍵∘C){(⍺⍺ F ⍬).f ⍵}))}


}
```

APL – Stateless!

```
cantor←{
  C←{(f: (ω.f){ω=0:ωω ⋄ αα ω-1}α)}
  F←{b←αα P 0 ⋄ αα b:b ⋄ αα P 1}
  P←{ω C (P: P ⋄ C: C ⋄ F: F
      f: (αα∘(ω∘C){(αα F θ).f ω}))}
  A←{αα(~∘αα F)ω}

}
```

APL – Stateless!

```
cantor←{
  C←{(f: (ω.f){ω=0:ωω ⋄ αα ω-1}α)}
  F←{b←αα P 0 ⋄ αα b:b ⋄ αα P 1}
  P←{ω C (P: P ⋄ C: C ⋄ F: F
      f: (αα∘(ω∘C){(αα F ⍬).f ω}))}
  A←{αα(~∘αα F)ω}
  (αα≡ωω)A ω
}
```

APL

```
      ({⊃∧/ω.f¨1 3 5} Cantor {(ω.f 1)∧(ω.f 3)∧(ω.f 6)})⍬
0
      ({⊃∧/ω.f¨1 3 5} Cantor {(ω.f 1)∧(ω.f 3)∧(ω.f 5)})⍬
1
      ({3=+/ω.f¨⌽⍳5} Cantor {3=+/ω.f¨⍳5})⍬
1
      ({3=+/ω.f¨⌽⍳5} Cantor {3=+/ω.f¨⍳4})⍬
0
```

Formal power series 101

- Here: considered independently from any notion of convergence and can be manipulated with the usual algebraic operations.
- Consider the following power series representing the sine and cosine functions. They will serve as illustratory examples

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

APL representation

- Take the coefficients of the powers of ω. A power series is a dfn $\boldsymbol{N} \to \boldsymbol{R}$, i.e. a mapping from the term number to the coefficient.
- Example:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

```
cos←{2|⍵:0
      neg←2|0.5×⍵
      neg:÷-!⍵ ⋄ ÷!⍵}
```

```
      cos¨⍳6
1 0 ¯0.5 0 0.041666667
```

Elementary operations

```
neg←{-αα ω}
add←{(αα+ωω) ω}
sub←{(αα-ωω) ω}
const←{ω<≢,α:ω⊃,α⋄0}
cons←{ω<≢,α:ω⊃,α⋄αα ω-≢,α}
```

$$\left(\sum_{i=0}^{\infty} a_i x^i\right) \cdot \left(\sum_{j=0}^{\infty} b_j x^j\right) = \sum_{k=0}^{\infty} c_k x^k$$

$$c_k = \sum_{l=0}^{k} a_l b_{k-l}$$

```
mul←{l←⍳1+ω⋄+/(αα¨l)×(ωω¨ω-l)}
```

Reciprocals

```
recip←{0=αα 0:⎕SIGNAL 8
       ω=0:÷αα 0
       z←1+⍳ω
       (-÷αα 0)×+/(αα¨z)×(αα ∇∇)¨ω-z}
```

$$b_0 = \frac{1}{a_0},$$

$$b_n = -\frac{1}{a_0}\sum_{i=1}^{n} a_i b_{n-i}, \quad n \geq 1.$$

Consequence of the Faà di Bruno's formula.

Composition

```
jot←{0≠ωω 0:⎕SIGNAL 8
     ω=0:αα 0
     ((ωω 1∘+)mul((αα 1∘+)jot ωω))ω-1}
```

~ Douglas McIlroy, Functional Pearls

Integration and derivatives

```
derv←{(αα ω+1)×ω+1}
int←{α←0⋄ω=0:α⋄(αα ω-1)÷ω}
```

$$\int_0^z \sum_{i=0}^{\infty} a_i x^i \, dx = \sum_{i=0}^{\infty} a_i \frac{x^{i+1}}{i+1}$$

```
      (¯1∘(sin int)¨ ≡ (cos neg)¨)⍳10
1
```

Surprisingly compact.

```
      exp←{÷!ω}
      log1p←{(¯1∘* int) ω}

      tan←sin div cos
      .5⊥⌽tan¨⍳20
0.5463024897923674093178175472236696
      3○.5
0.5463024898437905132551794657802855
```

Recursive definitions

```
    my_exp←{1(my_exp int)ω}
    my_sin←{(my_cos int) ω}
    my_cos←{((1∘const) sub (my_sin int))ω}
    (cos¨⍳5) ≡ (my_cos¨⍳5)
1
    (sin¨⍳5) ≡ (my_sin¨⍳5)
1
    (exp¨⍳5) ≡ (my_exp¨⍳5)
1
```

Why?

- Many mathematical functions of particular interest can be written as formal power series!
- Demonstrating or proving analytic results through purely algebraic means.
- Elegant, instructive examples of functional programming.
- APL: A versatile language which caters to pragmatics and dreamers.

Thank you for your attention!

- Reach out to me! kspalaiologos@gmail.com
- My blog: https://palaiologos.rocks/
- My PGP key: C868 F0B6 DE38 409D
- Read the paper with full source code!