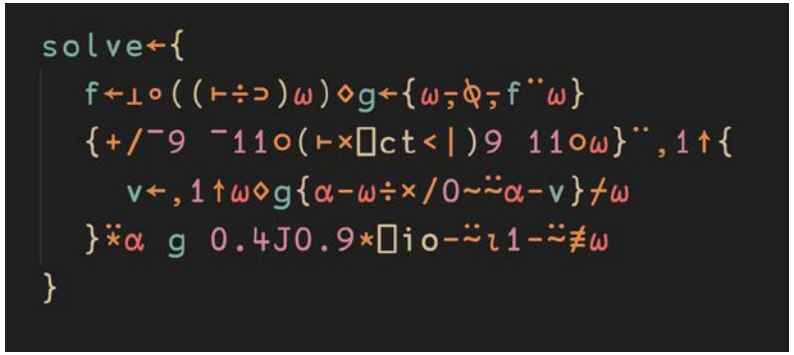
An Implementation of APL Array Notation

Who am I?

- Student @ Saarland University
- Intern @ Dyalog Ltd.
- A C programmer enamored with APL's applications to mathematics



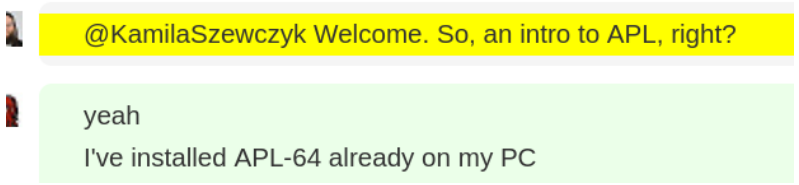
Blog: https://palaiologos.rocks

What if APL was made for solving abstract CS and mathematics problems?

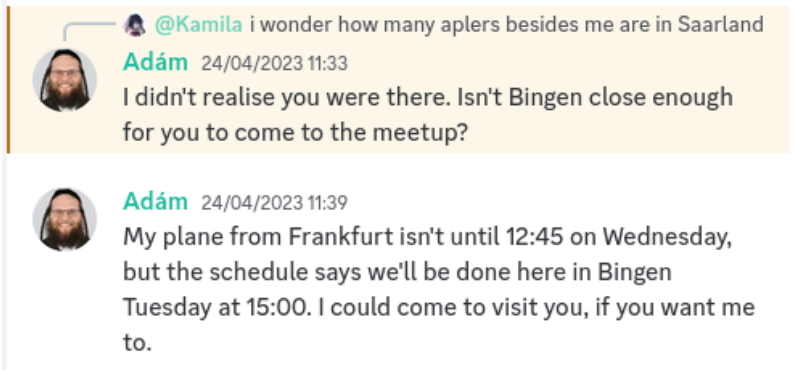
Who am I?

Learned about Array Programming in 2018 from IRC (#jsoftware).

Summer of 2020:



Spring of 2023:



My time at Dyalog

- **APL Array notation:**
  - 62582⌶: Deserialise Array Notation to an APL array/object
  - 62583⌶: Prettify Array Notation
  - Found a problem with the formal specification
- ⍁DIFF: Automatic Differentiation.
- ⍥: Reverse Compose.
- Speed up: ⍳⍣¯1, N∘⊥⍣¯1⊢M
- Iverson's monadic dot product (e.g. -.× D computes the determinant)
- Obverse (f⍨g where f has inverse of g)
- Monadic ∨/∧ (demote / promote)
- More...

Array Notation?

- Implemented by BQN.
- Parentheses and brackets may contain many expressions
- Parentheses may be empty!
- Three basic constructs.

```
(1◇2◇'a'◇⊢∘-\⍳5◇'abc') ≡ 1 2 'a' (1 ¯2 3 ¯4 5) 'abc'
```

(each expression becomes an element in a new vector)

```
[[1 2 ◇ 3 4] ◇ [5 6 ◇ 7 8]] ≡ 2 2 2⍴⍳8
```

(each expression becomes a major cell (of rank≥1) in a new array)

```
(name: 'Kamila' ◇ birthday: '09-07-2004')
```

(namespace syntax)

Why Array Notation?

- **Dyalog has been slowly moving away from workspaces and towards text-based source management.**
  - Easy to use SCM tools to manage your code.
  - Easy to modify and inspect the code.
  - …
- It is now possible to serialise and deserialise complex structures (imagine]Repr on steroids).
- It is faster than just executing APL, allows for a reasonably formatted multiline array notation and ultimately makes teaching and using APL easier.

What makes the Array Notation fast?

- First and foremost: It is statically parseable!

```
                    n gets: a, b and c stranded together?
n←a  b  c           n gets: a and c applied to b?
                    n gets: c applied to b, and then applied to a?
```

Meanwhile: we know what (a⋄b⋄c) is!

(Each expression becomes an element in a new vector. This is always a strand.)

What makes the Array Notation fast?

- The "fast execute" mechanism: Because the array notation is a subset of APL, we can assume/scan for certain things to determine the value of the expression without executing it.
- E.g.: Use 62582I to evaluate constants:

```
x←⍕÷?⍨1000000⋄y←⍕{'''',''''',⍨⎕a[(?10)↑?⍨10]}¨⍳2000⋄z←x,y,x,y,x,y

  cmpx '62582Ix' '±x'
62582Ix → 2.5E¯3 |   0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
±x      → 3.3E¯3 | +31% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
  cmpx '62582Iy' '±y'
62582Iy → 1.9E¯4 |    0% ⎕
±y      → 5.1E¯3 | +2650% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
  cmpx '62582Iz' '±z'
62582Iz → 8.8E¯3 |    0% ⎕
±z      → 2.0E¯1 | +2221% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

User Story

- Scenario: Over 40MB of real-world text and numeric data in Dyalog component files stored as arrays, namespaces, etc...
- Problem: One must be able to periodically find differences between old and new versions of the database and make it accessible to software that is not Dyalog APL.
- Possible solution: export as JSON? Develop a custom format?
- Caveat:

```
DOMAIN ERROR: the right argument cannot be converted
      ⎕JSON 2 2⍴4
      ^
```

User Story

- Actual solution: Just use the **APL Array Notation**!
- **Issue: Current implementation in ⎕SE is problematic:**
  - Poor performance characteristics
  - Not always correct
  - Does not handle certain edge cases
  - …
- Solution: A fast deserialiser and serialiser for the Array Notation written in C and integrated into the interpreter.

User Story

```
      cmpx '⎕SE.Dyalog.Array.Deserialise data'
  8.9E0   ⍝ 8s 900ms
      cmpx '62582⌶data'
  5.6E¯1  ⍝ 560ms
      wsreq '⎕SE.Dyalog.Array.Deserialise data'
557119736 ⍝ 557MB!
      wsreq '62582⌶data'
31706960  ⍝ 31MB
      ⎕size 'data'
23085216  ⍝ 23MB
```

User Story

- Not so simple: One needs to serialise the Array Notation first.

> We have installed Dyalog on a powerful machine and set MAXWS to 100G. The input array was 5MB, and the output file 9MB. ⎕SE's Serialiser took 55 seconds to run. So, that needed about 16GB of workspace...

- **The ⎕SE Serialiser is far from perfect, but the task it's performing is remarkably complex:**
  - When do we use APL strands - a b c - and when do we use the Array Notation syntax - (a⋄b⋄c)?
  - How do we format the resulting array notation?
  - How to represent tricky APL objects (tacit functions, dfns, scripted namespaces, tradfns, etc...)?
  - Due to data loss bugs in the past, the result is always cross-checked.

A dive into ⎕SE

Turning APL objects into expressions that result in them is difficult.

```
      f←'+'∘-  ⋄  g←+∘-
      f
+∘-
      g
+∘-
      ⍵ ¯\_(ツ)_/¯
```

Surprisingly: All the complex logic that handles this is **not** the bottleneck!

> Use the right tool for the job

To no surprise: APL is the right tool to write a serialiser. How certain data is represented internally by the C code is very different to how it appears to the APL programmer.

However: APL is **not** the right tool to write a formatter/prettifier!

A dive into ⍋SE

- After replacing the formatter in the serialiser code with the 62583 I-beam, the memory usage went down by more than an order of magnitude. The speed was doubled.
- C: Inherently scalar, rarely overcomputes. Formatting an Array Notation string does not call for array logic and is inherently a serial problem.
- APL: Array-oriented, often overcomputes. Processing performed as many small steps spanning the whole string.

Briefly about □DIFF

- Based on Taylor series, samples a few points around x+Δh.

$$\frac{\partial^{(1)} f}{\partial x^{(1)}} \approx \frac{f(x-2h) - 8f(x-1h) + 8f(x+1h) - f(x+2h)}{12h}$$

- Implementation detail: more points are being considered increasing accuracy but worsening the behaviour around singularities.
- More accurate than the central difference method (definition of derivative).

Briefly about □DIFF

```
      ((1∘○)□diff - 2∘○) 0.3 ⍟ □FR 645
1.950931461E¯8
      ((1○○)□diff - 2○○) 0.3 ⍟ □FR 1287
7.33843311E¯26
```

Future ideas:

- Dual numbers (hypercomplex number system, a + bε where ε^2 = 0 and ε ≠ 0).
- Complex derivatives.
- Numerical integration (Tanh-Sinh quadrature by default, Gauss-Legendre quadrature for smooth integrands)

Briefly about DIFF: Dual numbers

$$
\begin{aligned}
\frac{a+b\varepsilon}{c+d\varepsilon} &= \frac{(a+b\varepsilon)(c-d\varepsilon)}{(c+d\varepsilon)(c-d\varepsilon)} \\
&= \frac{ac-ad\varepsilon+bc\varepsilon-bd\varepsilon^2}{c^2+cd\varepsilon-cd\varepsilon-d^2\varepsilon^2} \\
&= \frac{ac-ad\varepsilon+bc\varepsilon-0}{c^2-0} \\
&= \frac{ac+\varepsilon(bc-ad)}{c^2} \\
&= \frac{a}{c}+\frac{bc-ad}{c^2}\varepsilon
\end{aligned}
$$

$$
h(x) = \frac{f(x)}{g(x)}
$$

$$
h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}.
$$

Crucial: a = f(x), b = f'(x), c = g(x), d = g'(x)

Briefly about □DIFF: Complex derivatives

Many ideas:

- Frechet derivative.
- Complex Finite difference stencils.
- …

Simple proof of concept: Use Cauchy-Riemann equations!
(Complex Variables with Applications, Jeremy Orloff, MIT)

Briefly about ⍁DIFF: CR equations

First step: f(x + iy) = u(x, y) + iv(x, y)

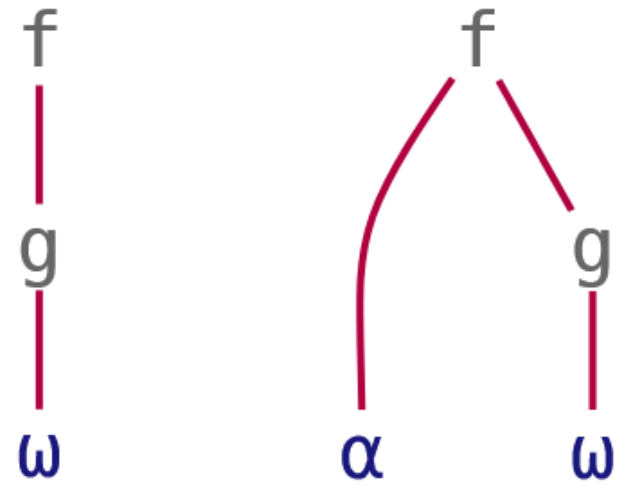
Definition: *f is complex differentiable at a complex point if and only if the partial derivatives of u and v satisfy the Cauchy-Riemann equations at that point.*

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x},$$

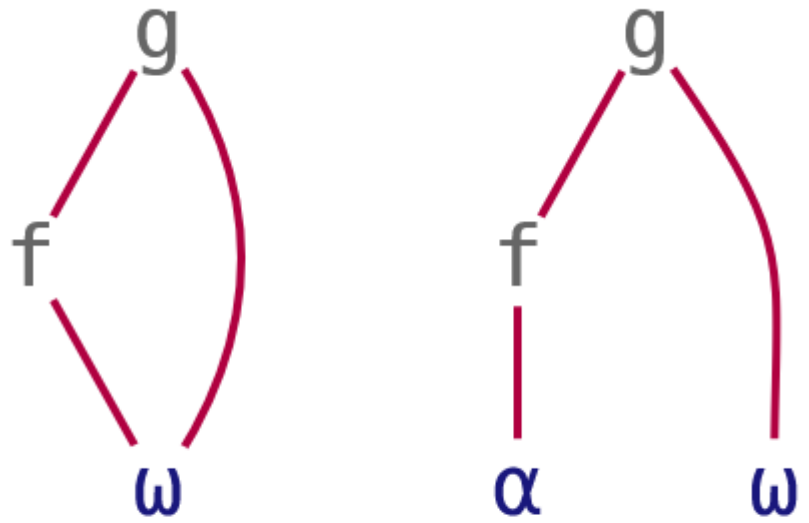
```
cmpxdiff←{
      f←⍺⍺ ⋄ re←9○⍵ ⋄ im←0J1×11○⍵
      dudx←{9○f ⍵+im}⍁DIFF re
      dvdx←{11○f ⍵+im}⍁DIFF re
      dudx+0J1×dvdx
}
```

Briefly about ⍥




```
f∘g ω      α f∘g ω
```

```
(f∘g) ω ⇔   f (g ω)
α (f∘g) ω ⇔ α f (g ω)
```

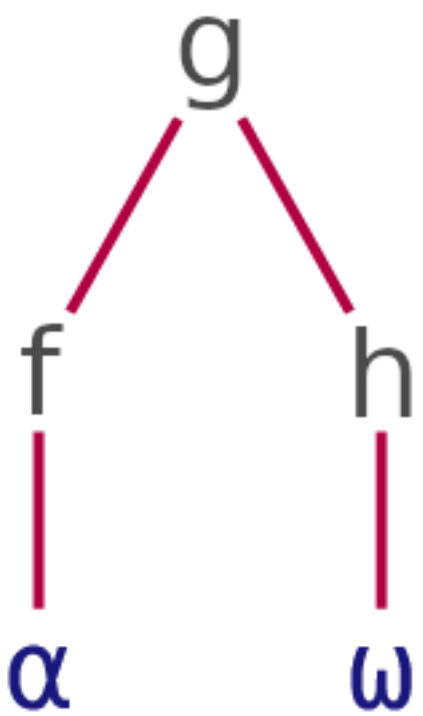



```
f⍥g ω      α f⍥g ω
```

```
(f⍥g) ω ⇔ (f ω) g ω
α (f⍥g) ω ⇔ (f α) g ω
```

Briefly about ⍥

```
α (f⍥g∘h) ω
```




```
Dyalog APL/S-64 Version 20.0.47746
Serial number: 201845
Wed Sep 27 20:13:12 2023
      5 ⍳⍥×∘| 5 ¯8 ¯2 ¯5 3
5 16 6 20 15
      ⍝ I don't like these:
      5 (⍳⍤⊣×|⍤⊢) 5 ¯8 ¯2 ¯5 3
5 16 6 20 15
      5 ×⍨∘⍳⍨∘| 5 ¯8 ¯2 ¯5 3
5 16 6 20 15
```

Thoughts? Questions?

```
z←{(⍦'(',')⍤¯1',⍨((⊂'+⊢×')(1↓⍬,,⍤0)⍤⍥⍵1↓⍵),⍤∈'+' (⍤⊃⍵) '×⊢')0}

z 1 2 3 4 5 6          (6+⊢×5+⊢×4+⊢×3+⊢×2+1×⊢) ¯1.491797988
   ¯1.491797988           1.580688469E¯9
```