The Power of Function Composition

Conor Hoekstra
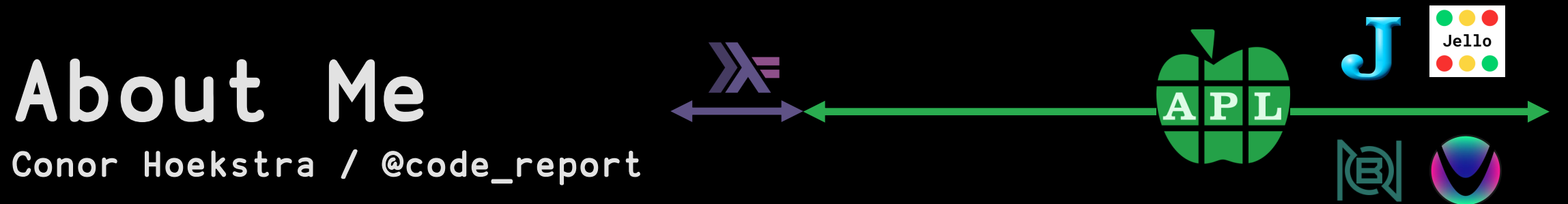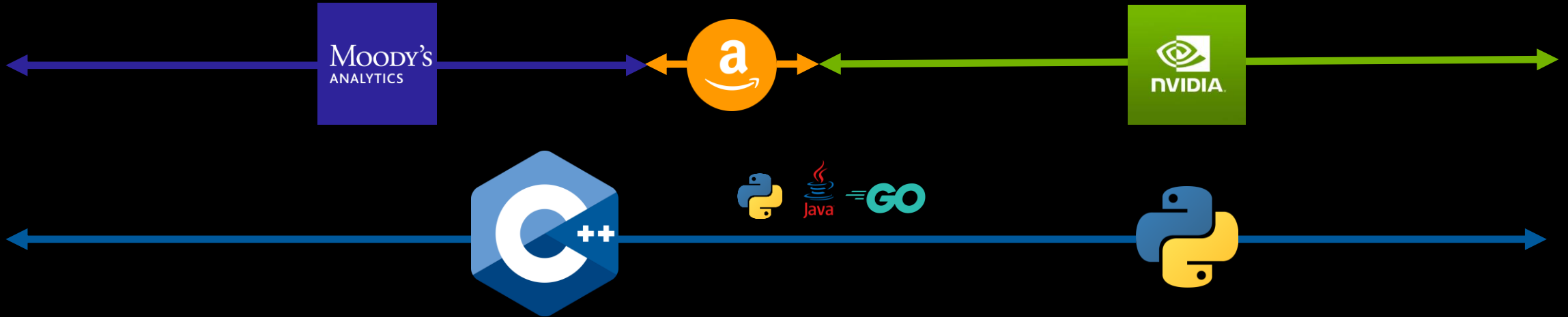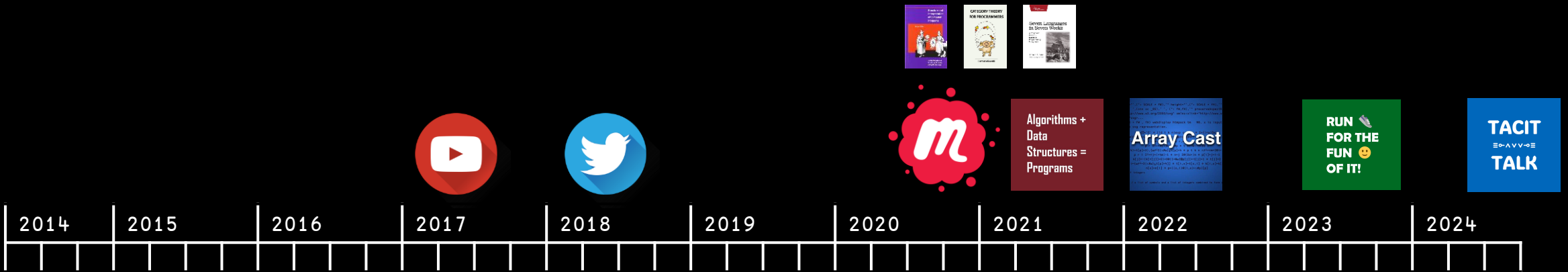
code_report | codereport

About Me

Conor Hoekstra / @code_report

351 Videos   40 (28) Talks

Algorithms +
Data
Structures =
Programs

Array Cast

TACIT
Ξ∘∧∨∨∘Ξ
TALK

RUN 👟
FOR THE
FUN 🙂
OF IT!

202 Episodes
@adspthepodcast

89 Episodes
@arraycast

5 Episodes
@codereport

20 Episodes
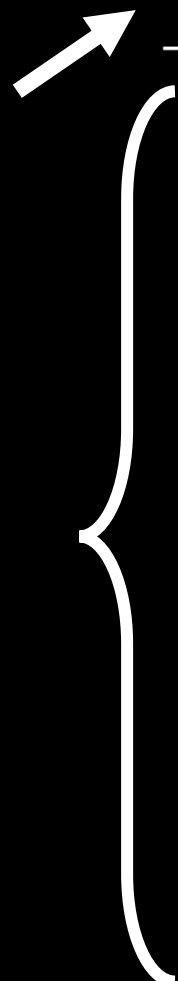@conorhoekstra

https://github.com/codereport/Content

Function Composition

Function Composition

1. Operators
2. Functions
3. Trains
4. Chains
5. Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	≈	≈	~	˜	`	.	join
C	≈	≈	~	˜	@	:	flip
B	∘∘̈ö	∘̈	@:&:	∘○	*	*	.
B_1	∘̈	∘̈	@:	∘	*	*	:.
S		∘	*	∘⊸		*	ap / <*>
Σ		∘̱		∘̸	*	*	=<<
D	∘	∘	*	∘		*	
Δ		∘̱		⊸∘	*	*	
Ψ	∘̈	∘̈	&:	○	*	∩	on
D_2		a∘̱b∘c		a⊸b⊸c		∩	
Φ	*	a≪b≫c	*	*	*	∩	liftA2
Φ_1	*	a≪b≫c	*	*	*	⊃	

Operators   Functions   Trains   Chains   Stacks*

combinator: a function that deals only in its arguments

CL combinator: a combinator from Combinatory Logic

FOF: a combinator that <u>only</u> consumes AND produces functions

pure function: same input = same output / no side effects

HOF: consumes OR produces a function

pure function

combinator

pure function   combinator

pure function

HOF

combinator

CL (SKI)
combinator

combinator: a function that deals only in its arguments
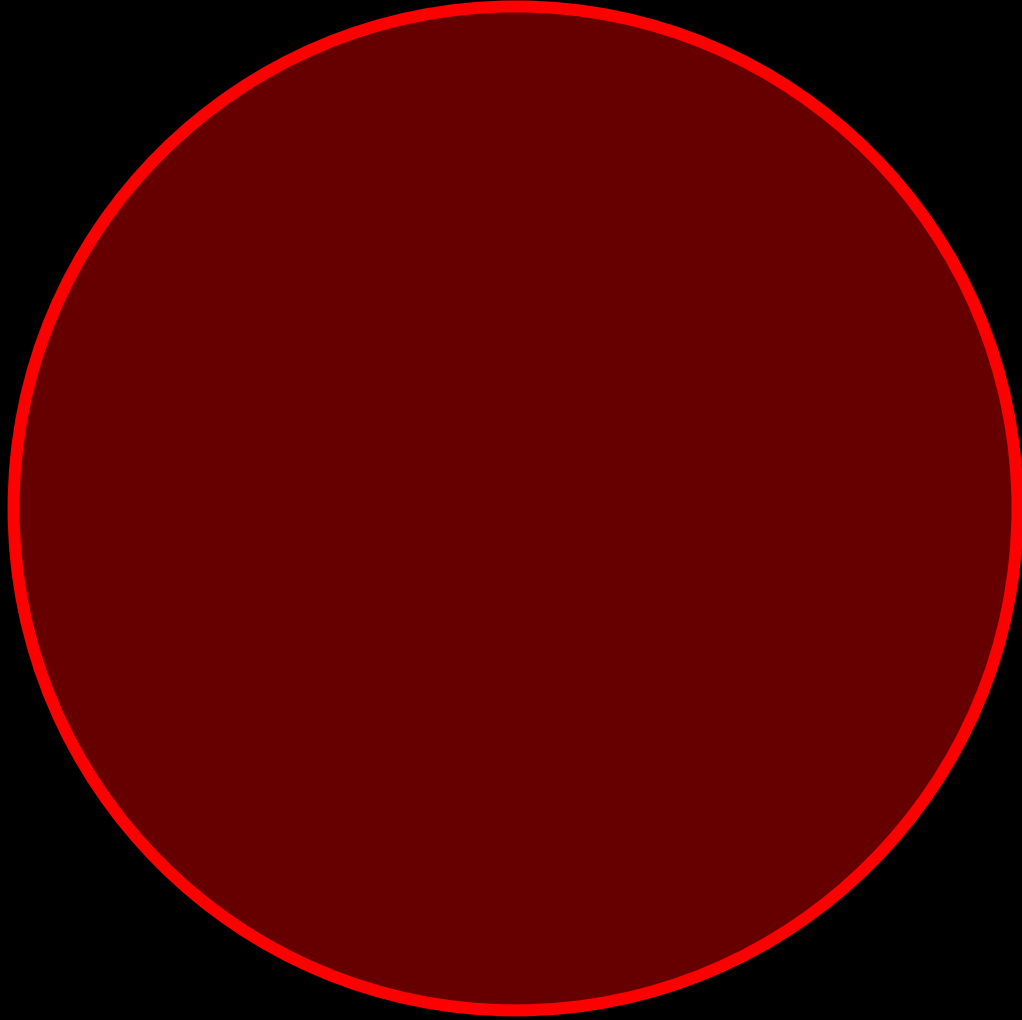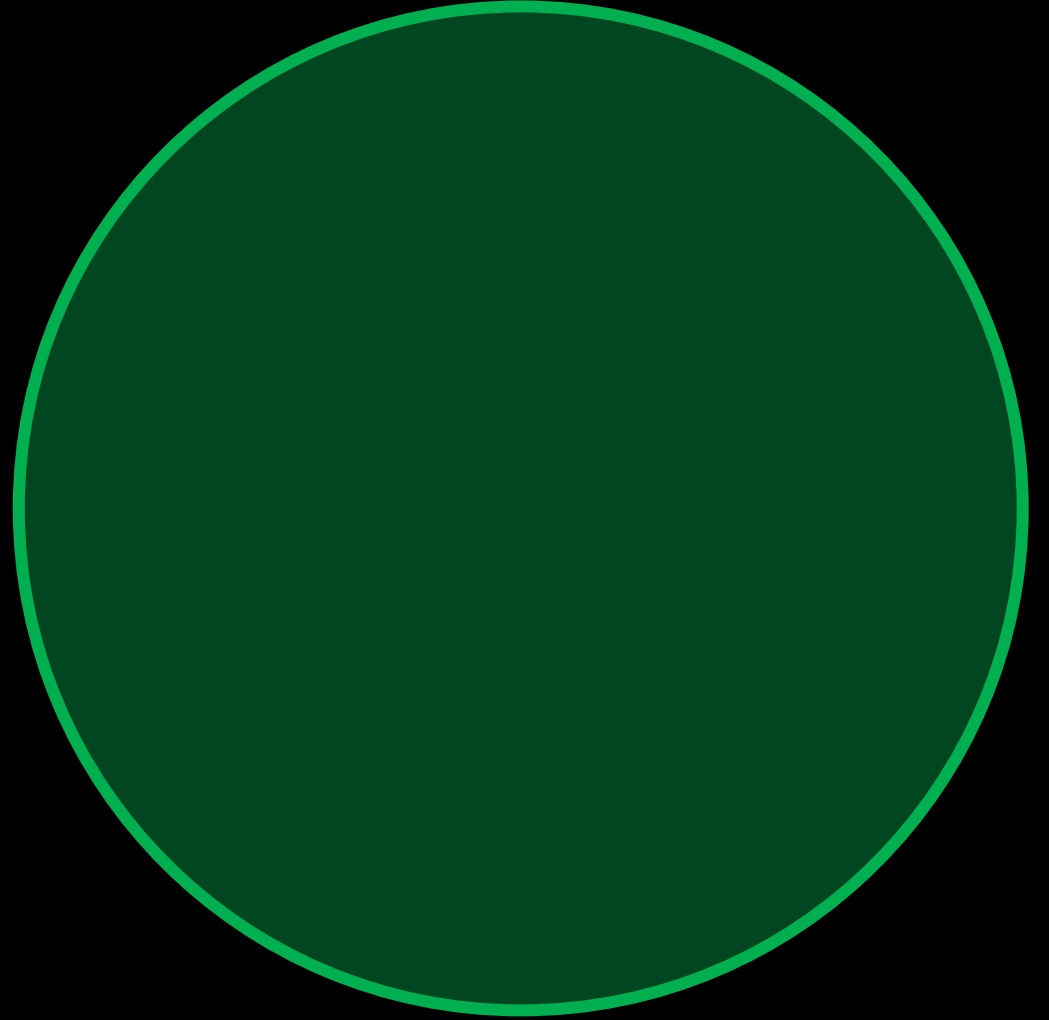
CL combinator: a combinator from Combinatory Logic

FOF: a combinator that <u>only</u> consumes AND produces functions

pure function: same input = same output / no side effects
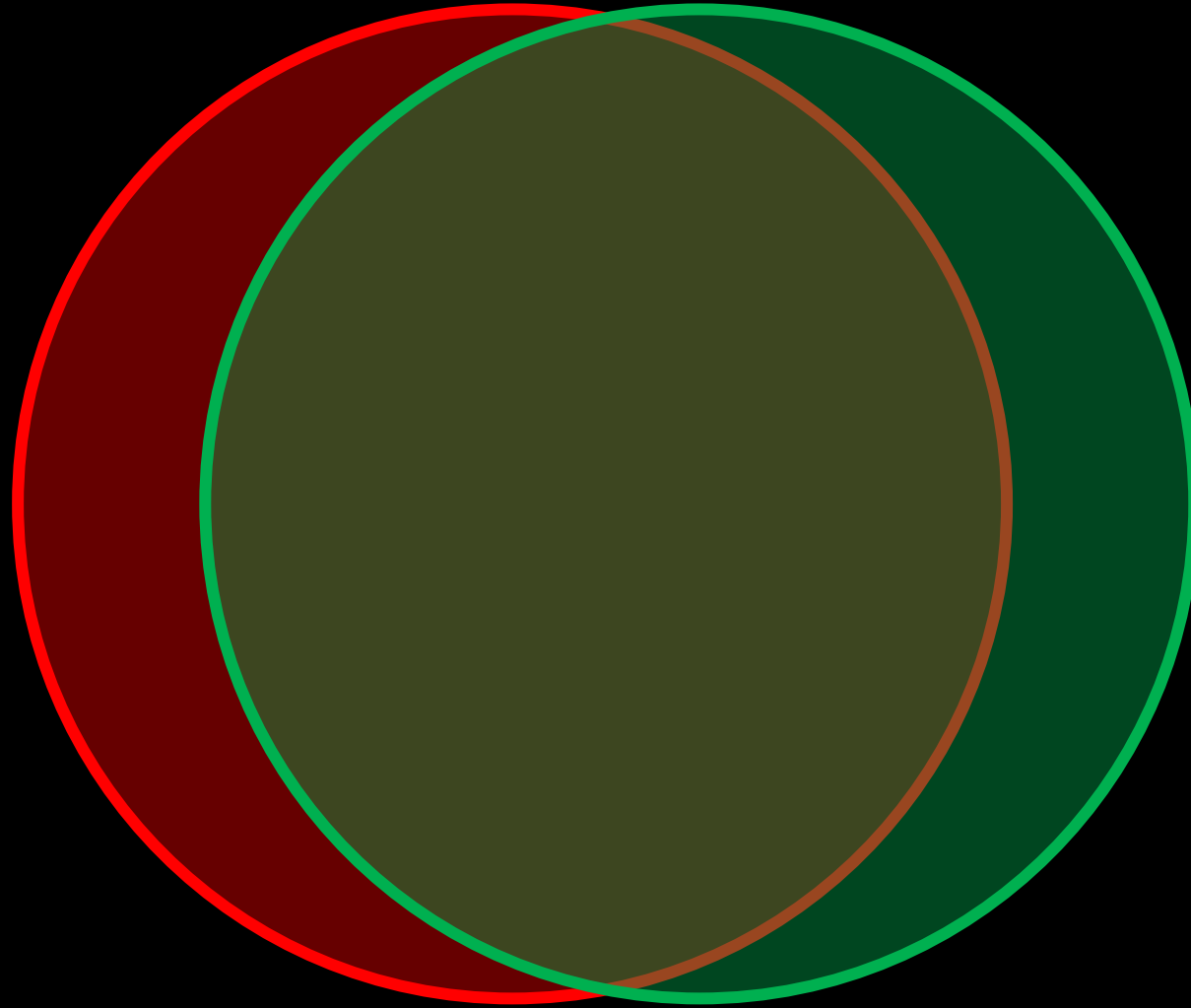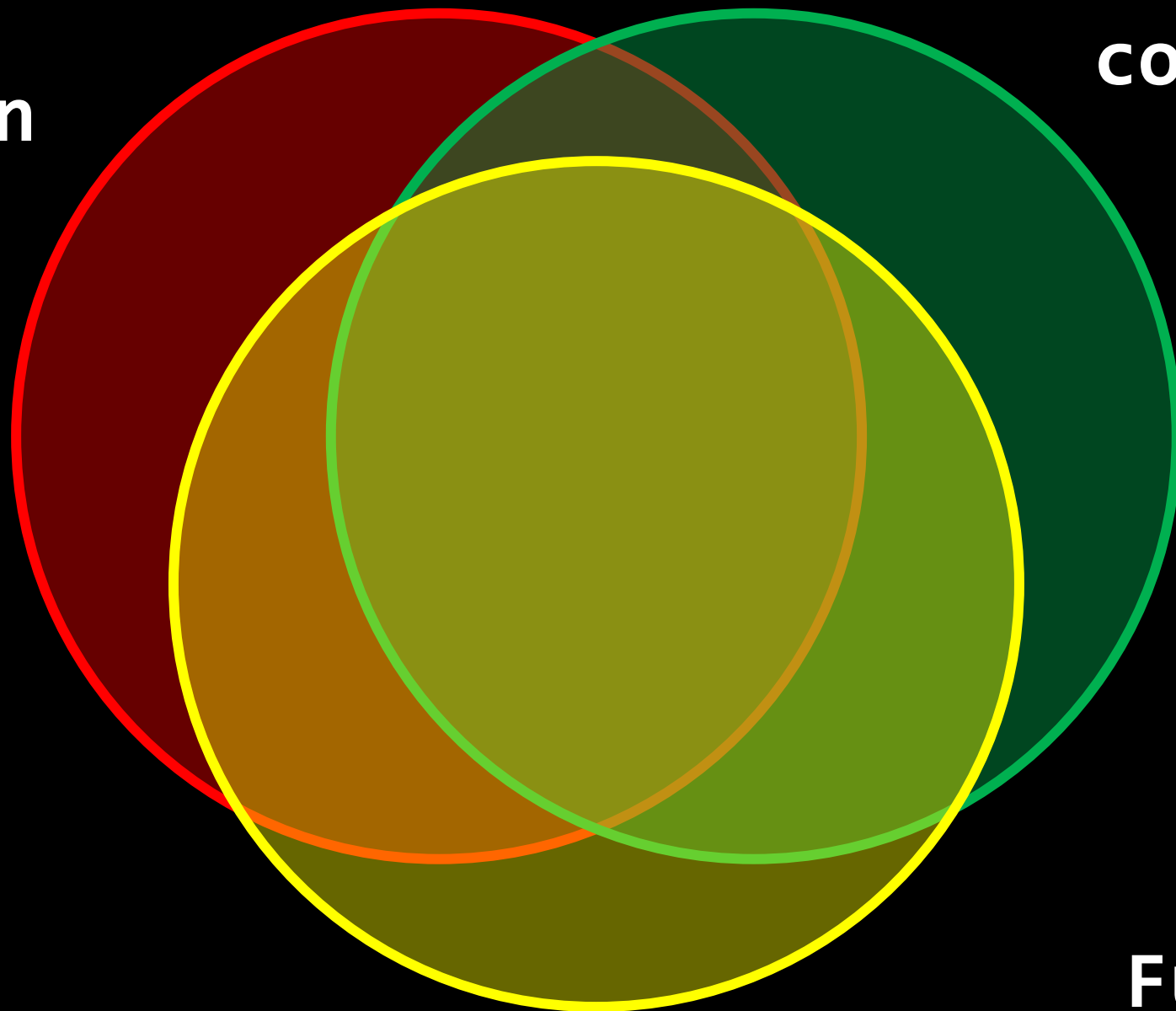
HOF: consumes OR produces a function

Combinators

THE ELEMENTARY COMBINATORS

Combinator	Elementary Name
I	Elementary Identificator
C	Elementary Permutator
W	Elementary Duplicator
B	Elementary Compositor
K	Elementary Cancellator

STUDIES IN LOGIC
AND THE
FOUNDATIONS OF MATHEMATICS
L. E. J. BROUWER / E. W. BETH / A. HEYTING
EDITORS

Combinatory Logic

VOLUME I

HASKELL B. CURRY
ROBERT FEYS

NORTH-HOLLAND PUBLISHING COMPANY
AMSTERDAM

```python
def i(x):
    return x
```

```python
def k(x, y):
    return x
```
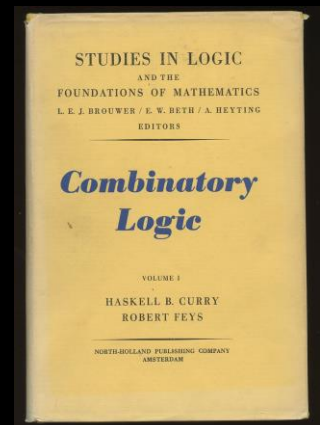
```python
def w(f):
    return lambda x: f(x, x)
```

[[digression]]

Conor Hoekstra @code_report · Jan 8, 2022

Also, I apologize for my above average number of tweets 🐦 today, but this table of Greek/Latin words for describing function **arity** will be necessary for a future talk.

The Ê combinator is "tetradic"

Unary/Monadic
Binary/Dyadic
Ternary/Triadic
Quaternary/Tetradic

Terminology [edit]

Latinate names are commonly used for specific arities, primarily base cardinal numbers or ordinal numbers. For example, 1-ary is based o

x-ary	Arity (Latin based)	Adicity (Greek based)
0-ary	*Nullary* (from *nūllus*)	*Niladic*
1-ary	*Unary*	*Monadic*
2-ary	*Binary*	*Dyadic*
3-ary	*Ternary*	*Triadic*
4-ary	*Quaternary*	*Tetradic*

💬 6   🔁 2   ❤️ 46   📊   ⬆️
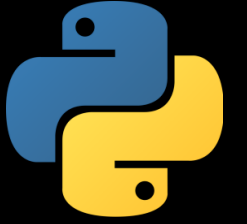
[[end of digression]]

```python
def w(f):
    return lambda x: f(x, x)
```

```python
def b(f, g):
    return lambda x: f(g(x))
```

```python
def c(f):
    return lambda x, y: f(y, x)
```

```python
def s(f, g):
    return lambda x: f(x, g(x))
```
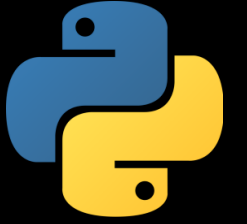
```python
def i  (x):        return x
def k  (x, y):     return x
def ki (x, y):     return y
def s  (f, g):     return lambda x:    f(x, g(x))
def b  (f, g):     return lambda x:    f(g(x))
def c  (f):        return lambda x, y: f(y, x)
def w  (f):        return lambda x:    f(x, x)
def d  (f, g):     return lambda x, y: f(x, g(y))
def b1 (f, g):     return lambda x, y: f(g(x, y))
def psi(f, g):     return lambda x, y: f(g(x), g(y))
def phi(f, g, h):  return lambda x:    g(f(x), h(x))
```

Example

Example
Special Array

3151. Special Array I

An array is considered **special** if every pair of its adjacent elements contains two numbers with different parity.

You are given an array of integers `nums`. Return `true` if `nums` is a **special** array, otherwise, return `false`.

4 3 1 6

| 4 | 3 | 3 | 1 | 1 | 6 |

| 0 | 1 | 1 | 1 | 1 | 0 |

| $0 \neq 1$ | $1 \neq 1$ | $1 \neq 0$ |

101

0
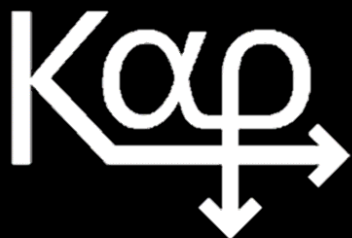
Python
Function Composition
✅ **Functions**
❌ **Operators**
❌ **Trains**
❌ **Chains**
❌ **Stacks**

```python
def isArraySpecial(nums):
    for i in range(len(nums) - 1):
        if nums[i] % 2 == nums[i + 1] % 2:
            return False
    return True
```

```python
def isArraySpecial(nums):
    for x, y in zip(nums, nums[1:]):
        if x % 2 == y % 2:
            return False
    return True
```

```python
def isArraySpecial(nums):
    return all(x % 2 != y % 2
               for x, y in zip(nums, nums[1:]))
```

```python
def isArraySpecial(nums):
    return all(x % 2 != y % 2
               for x, y in zip(nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```

```python
def psi(f, g):
    return lambda x, y: f(g(x), g(y))
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

Haskell

Function Composition

- ✅ **Functions**
- ✅ **Operators**
- ❌ **Trains**
- ❌ **Chains**
- ❌ **Stacks**

```haskell
isArraySpecial = foldl1 (&&)
               . ((zipWith (/=)) <*> tail)
               . map odd
```

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜		.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍤∘	⍤	@:&:	∘○	*	*	.
B_1	⍤	⍤	@:	∘	*	*	:.
S		∘	*	⊸		*	ap / <*>
Σ		⍛		⟜	*	*	=<<
D	∘	∘	*	⟜		*	
Δ		⍛		⊸	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D_2		a∘̲b∘c		a⊸b⊸c		⊓	
Φ	*	a≪b≫c	*	*	*	⊃	liftA2
Φ_1	*	a≪b≫c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

```haskell
isArraySpecial = foldl1 (&&)
               . ((zipWith (/=)) <*> tail)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)

isArraySpecial = foldl1 (&&)
               . mapAdjacent (/=)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)

isArraySpecial = and
               . mapAdjacent (/=)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)
import Data.Function (on)

isArraySpecial = and
               . mapAdjacent (on (/=) odd)
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne


def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(nums, psi(ne, odd)))
```

```python
from dovekie import odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```

Clojure
Function Composition
- ✅ **Functions**
- ✅ **Operators**
- ❌ **Trains**
- ❌ **Chains**
- ❌ **Stacks**

```clojure
(defn is-special-array [nums]
  (->> nums
       (partition 2 1)
       (map #(reduce not= %))
       (every? identity)))
```

```clojure
(defn is-special-array [nums]
  (->> nums
       (partition 2 1)
       (every? (fn [[a b]] not= a b))))
```
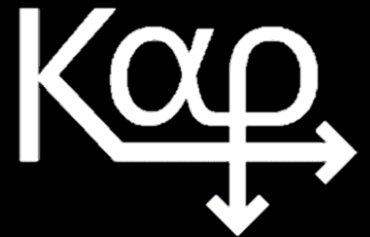
Table 4: 2 and 3-trains in APL, Kap, BQN and J.

Year	Language	2-Train	3-Train
1990	J	S and D	Φ and Φ_1
2014	Dyalog APL	B and B_1	Φ and Φ_1
2020	Kap	B and B_1	-
2020	BQN	B and B_1	Φ and Φ_1

```python
def i  (x):        return x
def k  (x, y):     return x
def ki (x, y):     return y
def s  (f, g):     return lambda x:    f(x, g(x))
def b  (f, g):     return lambda x:    f(g(x))
def c  (f):        return lambda x, y: f(y, x)
def w  (f):        return lambda x:    f(x, x)
def d  (f, g):     return lambda x, y: f(x, g(y))
def b1 (f, g):     return lambda x, y: f(g(x, y))
def psi(f, g):     return lambda x, y: f(g(x), g(y))
def phi(f, g, h):  return lambda x:    g(f(x), h(x))
```

```python
def s  (f, g):     return lambda x:    f(x, g(x))
def b  (f, g):     return lambda x:    f(g(x))

def d  (f, g):     return lambda x, y: f(x, g(y))
def b1 (f, g):     return lambda x, y: f(g(x, y))

def phi(f, g, h): return lambda x:    g(f(x), h(x))
```
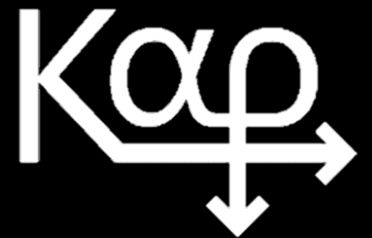
```python
def b  (f, g):  return lambda x:    f(g(x))
def b1 (f, g):  return lambda x, y: f(g(x, y))

def phi (f, g, h): return lambda x:    g(f(x), h(x))
def phi1(f, g, h): return lambda x, y: g(f(x, y), h(x, y))
```

Live Coding

Jelly

Function Composition

❌ **Functions**
✅ **Operators**
❌ **Trains**
✅ **Chains**
❌ **Stacks**

●●● Jello ●●●

> [4,1,6]

●●● Jello ●●●

> [4,1,6] :: odd?
                Ḃ
   Ḃ   [4,1,6] ➡ [0, 1, 0]

```
●●●  Jello  ●●●

> [4,1,6] :: odd? differ
                 Ḃ        Ď
   Ḃ      [4,1,6]  ➡  [0, 1, 0]
   ḂĎ     [4,1,6]  ➡  [1, 1]
```

●●● Jello ●●●

> [4,1,6] :: odd? differ all

	Ḃ	Ď	Ạ
Ḃ	[4,1,6] ➡	[0, 1, 0]	
ḂĎ	[4,1,6] ➡	[1, 1]	
ḂĎẠ	[4,1,6] ➡	1	

●●● Jello ●●●

> [4,1,6] :: odd? differ all

		Ḃ	Ď	Ạ
Ḃ	[4,1,6]	➡ [0, 1, 0]		
ḂĎ	[4,1,6]	➡ [1, 1]		
ḂĎẠ	[4,1,6]	➡ 1		

This is a 1-1-1 monadic chain (BB)

```
    ┌─┴─┐  ⋮
      B   ⋮
      ┌─┴─┐
        B
```

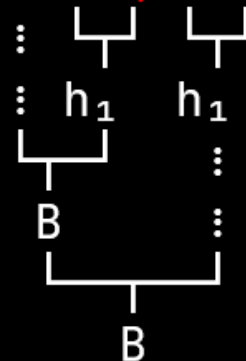●●● Jello ●●●

> [4,1,6] :: odd? ≠ prior and fold
            Ḃ   n   ṗ   a   /

    and fold can be replaced with all
      ☝️🧕 algorithm advisor 🧕☝️

    ≠ prior can be replaced with differ
      ☝️🧕 algorithm advisor 🧕☝️

    Ḃ   [4,1,6] ➡️ [0, 1, 0]
    Ḃnṗ   [4,1,6] ➡️ [1, 1]
    Ḃnṗa/ [4,1,6] ➡️ 1
    This is a 1-2-q-2-q monadic chain (BB)
              ⋮   ⌐⌐   ⌐⌐
              ⋮   h₁   h₁
                 ⌐⌐   ⋮
                 B   ⋮
                 ⌐⌐⌐⌐⌐⌐
                   B

Uiua
Function Composition
❌ **Functions**
✅ **Operators**
❌ **Trains**
❌ **Chains**
✅ **Stacks**

Kap

∧/2≠/2|

B B

Dyalog APL

∧/2≠/2|⊢

B φ φ

Uiua

/↧≡/≠⊞2⊿2

BQN

😈

∧´·≠´˘2↕2|⊢

B B φ φ

J

[: *./ 2~:/\ 2|[

B φ φ

Jello

odd? differ all

B B

	Python	Haskell	Clojure	APL	Kap	J	BQN	Jello	Uiua
Functions	✅	✅	✅	❌	❌	❌	❌	❌	❌
Operators	❌	✅	✅	✅	✅	✅	✅	✅	✅
Trains	❌	❌	❌	✅	✅	✅	✅	❌	❌
Chains	❌	❌	❌	❌	❌	❌	❌	✅	❌
Stacks	❌	❌	❌	❌	❌	❌	❌	❌	✅

In Conclusion

	Kap	`∧/2≠/2|`
	Dyalog APL	`∧/2≠/2|⊢`
	Uiua	`/↧≡/≠⊞2⊿2`
	BQN	`∧´·≠´˘2‡2|⊢`
	J	`[: *./ 2~:/\ 2|[`
	Jello	`odd? differ all`

NIKLAUS WIRTH

lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling

PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

Algorithms +

~~Data~~ Combinators =

~~Structures~~

~~Programs~~ Beautiful Code

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	\`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍤	⍤	@:&:	∘○	*	*	.
B₁	⍤	⍤	@:	∘	*	*	:.
S		∘	*	∘⟜		*	ap / <*>
Σ		⍛		∘⟜	*	*	=<<
D	∘	∘	*	∘○		*	
Δ		⍛		⟜∘	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D₂		a∘b∘c		a⊸b⟜c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
Φ₁	*	a«b»c	*	*	*	⊃	

Operators   **Functions**   **Trains**   **Chains**   **Stacks***

```python
from dovekie import odd
from operator import ne

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```

	Kap	∧/2≠/2|
	Dyalog APL	∧/2≠/2|⊢
	Uiua	/↧≡/≠⊞2⊿2
	BQN	∧´·≠´˘2‡2|⊢
	J	[: *./ 2~:/\ 2|[
	Jello	odd? differ all

Online REPLS

Language	Link
APL	https://tryapl.org/
Kap	https://kapdemo.dhsdevelopments.com/clientweb2/
BQN	https://bqnpad.mechanize.systems/
J	https://jsoftware.github.io/j-playground/bin/html2/
Uiua	https://www.uiua.dev/pad

www.combinatorylogic.com

Thank You

https://github.com/codereport/Content/Talks

Conor Hoekstra

🐦 code_report

▶️ codereport

Questions?

https://github.com/codereport/Content/Talks

Conor Hoekstra

code_report

codereport