DYALOG

APL Germany

Language Enhancements

Adám Brudzewsky

DYALOG

APL Germany

Language Enhancements

Indexing with nested vectors in APL

Ask Question

Asked 2 years, 3 months ago Modified 2 years, 2 months ago Viewed 132 times

3

I have a vector of vectors that contain some indices, and a character vector which I want to use them on.

```
A←(1 2 3)(3 2 1)
B←'ABC'
```

3

I have a vector of vectors that contain some

```
A←(1 2 3)(3 2 1)
B←'ABC'
```

I have tried:

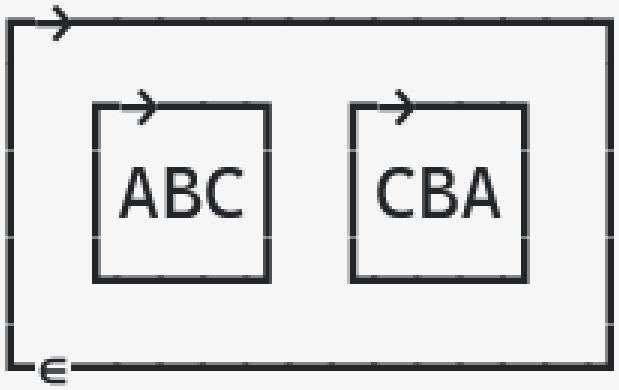
```
      B[A]
RANK ERROR
      B[A]
        ∧
```


```
      A□B
LENGTH ERROR
      A□B
        ^
```

and

```
      A□B
LENGTH ERROR
      A□¨B
        ^
```


I would like



to be returned, but if i need to find another way,

indexing apl

Indexing with Nested Vectors

3 Answers:

- `(⊂⍤1 A)[]⍤1⊂B`
- `{B[⍵]}⍤⍤⍤1 A`
- *Don't do that!*

Possibilities:

- `A[]⍤⍤⍤⍤⊂⊂B`
- `[]∘B⍤1⊂⍤1 A`
- `A[]⍨⍤⍤∘⊂⍨⍤⍤⊂B`

Indexing with Nested Vectors

3 Answers:

- `(⊂⍤1 A)[]⍤1⊂B`
- `{B[⍵]}⍤⍤⍤⍤1 A`
- *Don't do that!*

Possibilities:

- `A[]⍤⍤⍤⍤⊂⊂B`
- `[]∘B⍤1⊂⍤1 A`
- `A(⊂⍤⍤⊣[]⊢)⍤1⊂B`

Language Enhancements

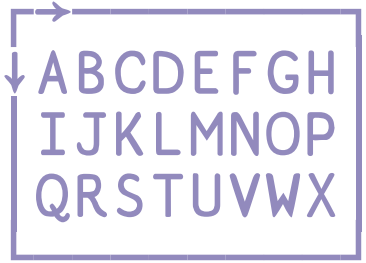
Core Language Enhancements

Data Transformation	X × Y	ϕ Y	X [] Y
Function Application	f ⊱	f ⍣ g	f ⍤ k
Function Composition	f ⍤ g	f ⍥ g	f ∘ g
Axis Manipulation	, Y	⍪ Y	X ⌽ Y

Data Transformation

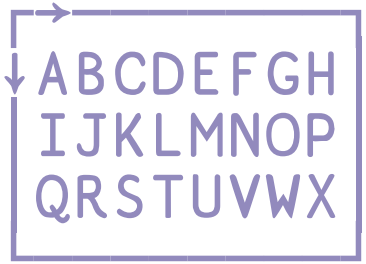
- Simple Indexing
  Choose Indexing
  Reach Indexing

```
      t←3 8⍴⎕A
      t[2;4]
L
      2 4⌷t
L
```



Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing




```
      t←3 8⍴⎕A

      t[2 1;4 1 7]
LIO
DAG

      (2 1)(4 1 7)⌷t
LIO
DAG
```

Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing


```
      p←8⍴p⌷A
      p[2]
B
      2⌷p
B
```

Data Transformation

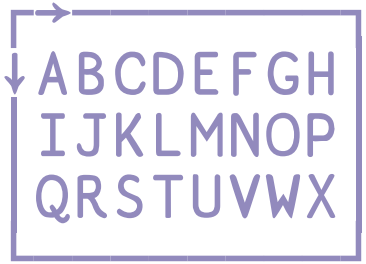
- Simple Indexing
  Choose Indexing
  Reach Indexing


```
      p←8⍴p⎕A
      p[2 1 7]
BAG
      2 1 7?p
BAG
```


Data Transformation

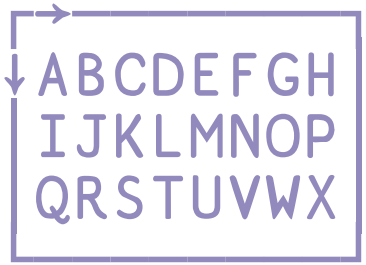
- Simple Indexing
- Choose Indexing
- Reach Indexing

```
t←3 8⍴⎕A
t[⊂1 8]
H
1 8⌷t
H
```



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

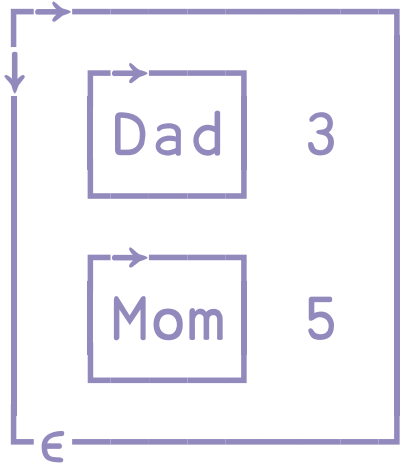



```
      t←3 8⍴⎕A
      t[(1 8)(2 7)]
HO
      (1 8)(2 7)⍰t
HO
```


Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

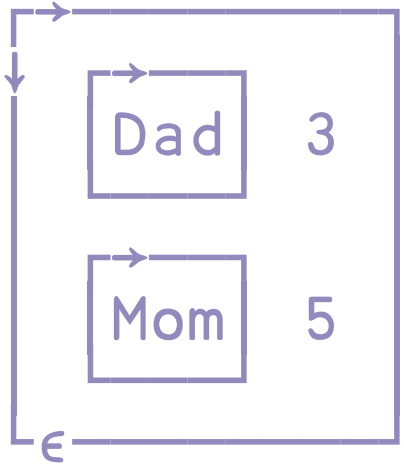
```
s←'Dad' 'Mom',⍪3 5
s[⊂(2 1)3]
m
(2 1)3⊃s
m
```



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

```
s←'Dad' 'Mom',⍪3 5
s[((2 1)3)((1 1)2)]
ma
((2 1)3)((1 1)2)⍰s
ma
```



Data Transformation

Data Transformation

Select

Indexing with Nested Vectors

3 Answers:

- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⌸A`
- *Don't do that!*

Possibilities:

- `A[]⌸⊂⊂B`
- `[]∘B¨⊂¨A`
- `A(⊂⍤⊣[]⊢)¨⊂B`

With Select `X⊇Y`:

- `A⊇¨⊂B`
- `⊇∘B¨A`

Data Transformation

X⊒Y Select

- Sort ← (⊂⍤⍋⌷⊢)
- Sorts ← ⌷⍨∘⊂∘⍋⍨ ⍝ "sort Y by X"
- Shuffle ← (⊂⍤?⍨∘≢⌷⊢)
- Grade ← ((⊂bounds∘⍳)⌷grades⍨)

Data Transformation

X⊒Y Select

- `Sort    ← (  ⍋⊒⊢)`
- `Sorts   ← ⊒⍨  ∘⍋⍨    ⍝ "sort Y by X"`
- `Shuffle ← (  ?⍨∘≢⊒⊢)`
- `Grade   ← (  bounds∘⍳ ⊒grades⍨)`

Data Transformation

X⊒Y Select

- Sort ← (⍋⊒⊢)
- Sorts ← ⊒⍨∘⍋⍨ ⍝ "sort Y by X"
- Shuffle ← (?⍨∘≠⊒⊢)
- Grade ← (bounds∘⍳⊒grades⍨)

Data Transformation

X⊒Y Select/Permute

- `Sort    ← (⍋⊒⊢)`
- `Sorts   ← ⊒⍨∘⍋⍨      ⍝ "sort Y by X"`
- `Shuffle ← (?⍨∘≠⊒⊢)`

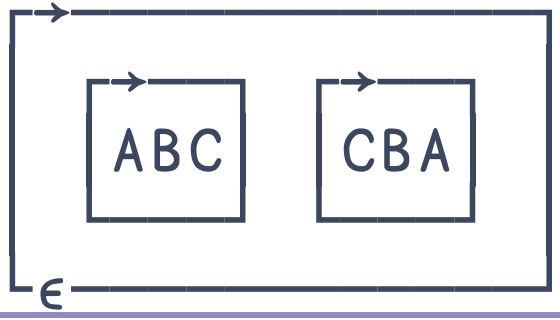
`Grade   ← (bounds∘⍳⊒grades⍨)`

Indexing with Nested Vectors

I have

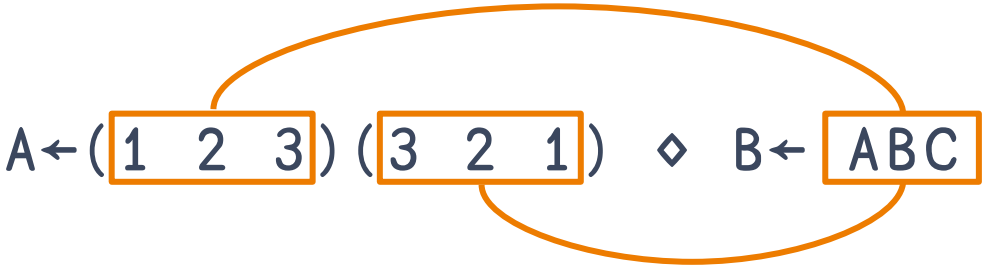
```
A←(1 2 3)(3 2 1) ⋄ B←'ABC'
```

I would like

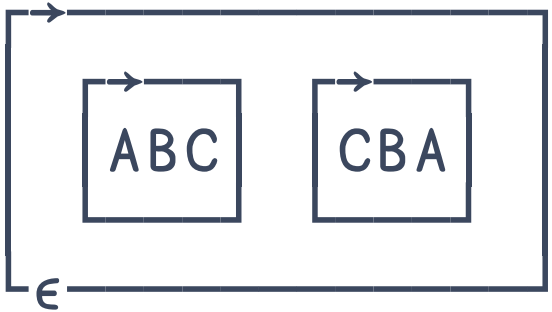


Indexing with Nested Vectors

I have

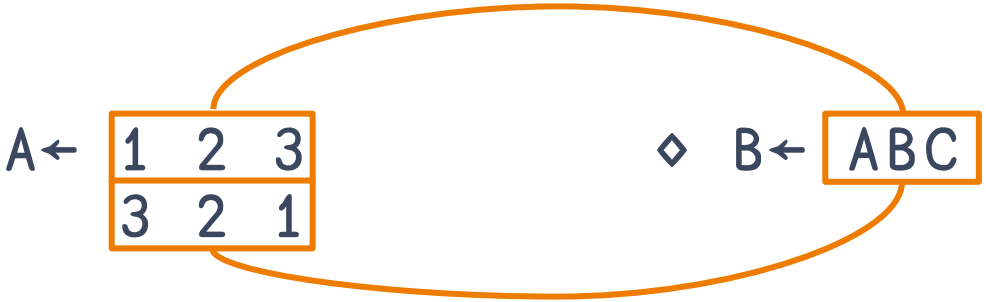


I would like

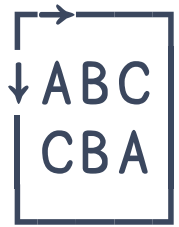


Indexing with Nested Vectors

I have

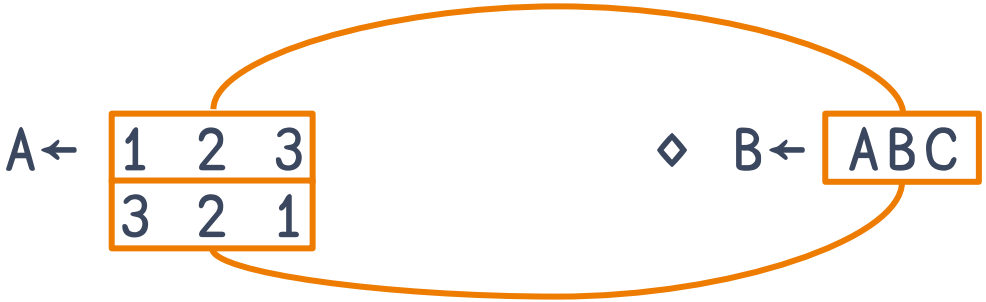


I would like

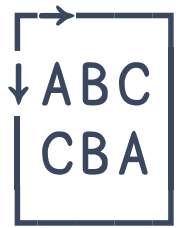


Indexing with Nested Vectors

I have



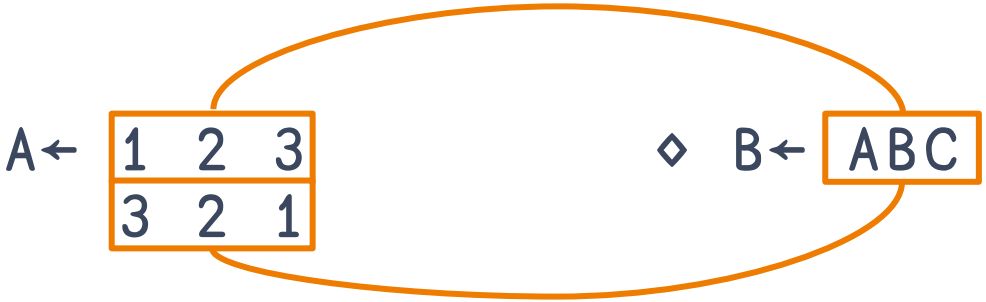
I would like



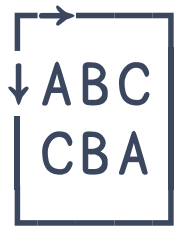

```
B[A]
```

Indexing with Nested Vectors

I have



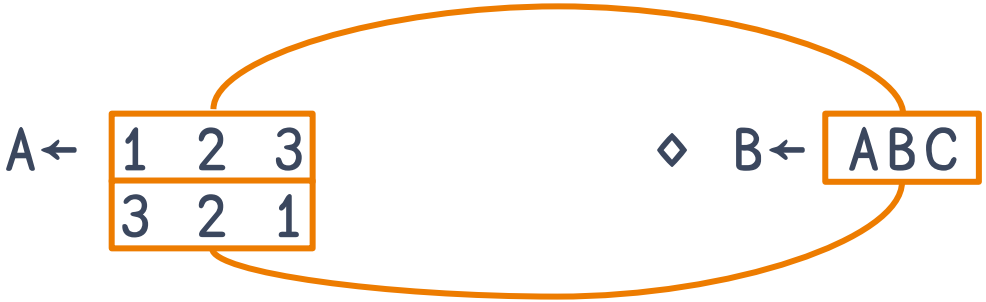
I would like



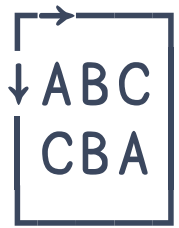
A ⊇ B

Indexing with Nested Vectors

I have



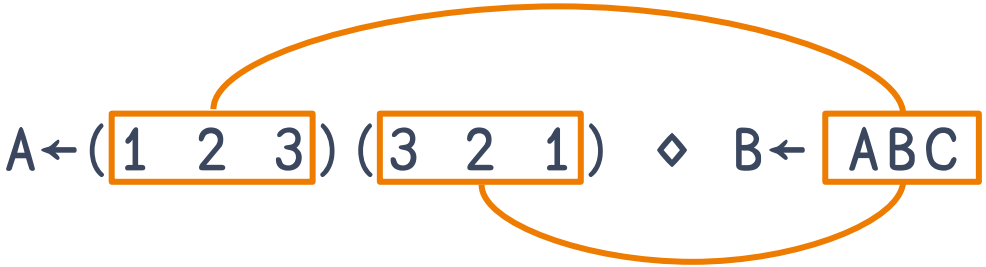
I would like



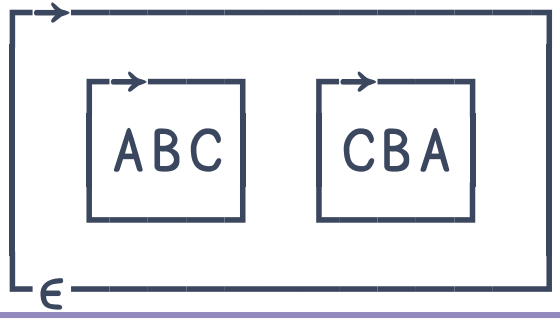

```
A(⍤⍥1)B
```

Function Application

I have



I would like




```
A(⍷?1)B
```

Function Application

Function Application

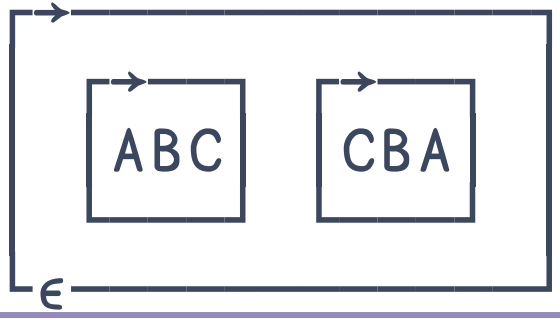
Depth

Function Application

I have

```
A←(1 2 3)(3 2 1) ⋄ B← ABC
```

I would like



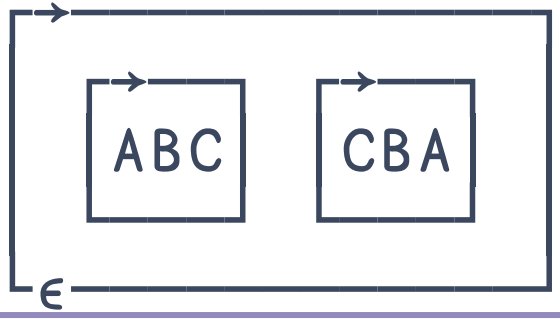

```
A(⍉⍤1)B
```

Function Application

I have

```
A←(1 2 3)(3 2 1) ⋄ B← ABC
```

I would like

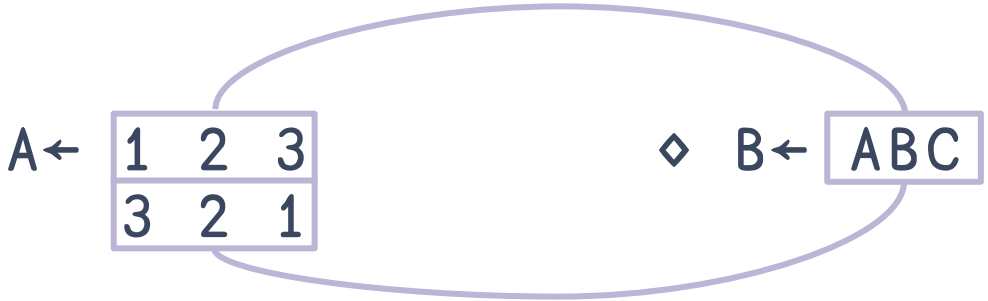



```
A(⍳⍤1)B
```

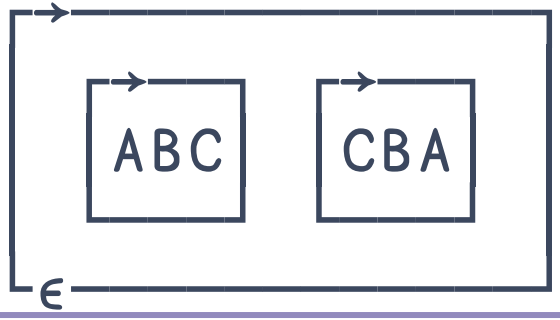
Watch this!

Function Application

I have



I would like

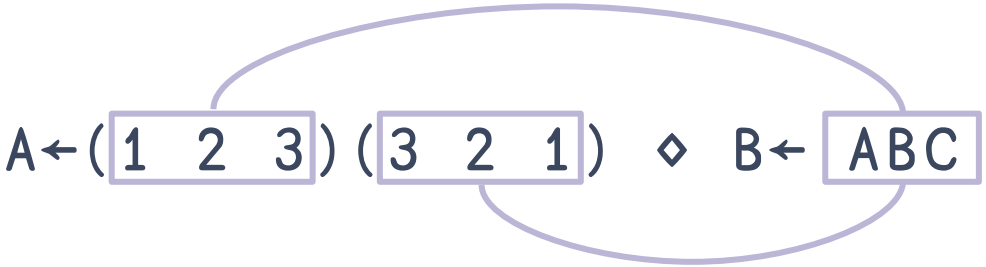


A(⊇⁚1)B

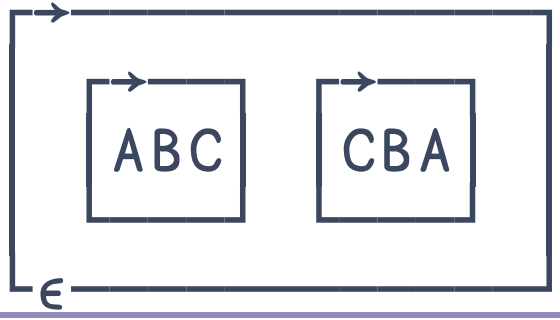
Watch this!

Function Application

I have



I would like



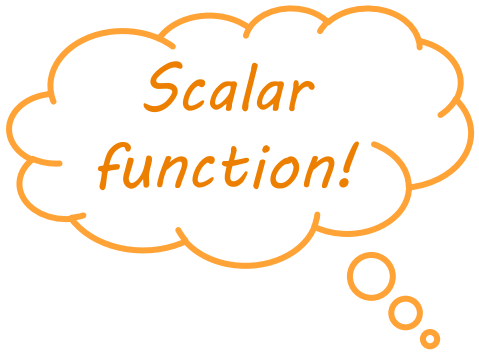
A(⍨⍤1)B

Watch this!

Function Application

```
Fp←!
Fp 4 (5 6)
```

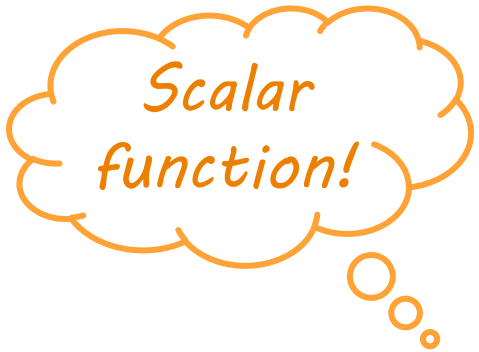
24	120 720



Function Application

```
      Fd←{×/⍳⍵}
      Fd 4 (5 6)
DOMAIN ERROR
Fd[0] Fd←{×/⍳⍵}
                ^
      Fs←{×/⍳⍵}⍤0
      Fs 4 (5 6)
```

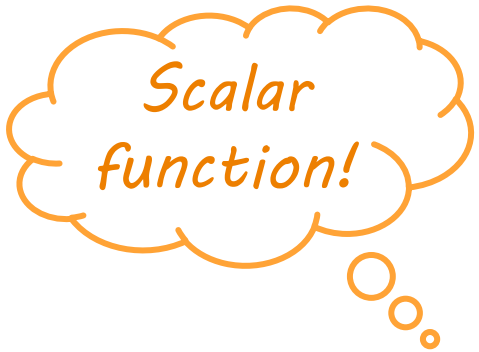
24	120 720



Function Application

```
      Fd←{×/⍳⍵}
      Fd 4 (5 6)
DOMAIN ERROR
Fd[0] Fd←{×/⍳⍵}
                ^
      Fs←{×/⍳⍵}⍤0
      Fs 4 (5 6)
```

24	120 720



Function Application

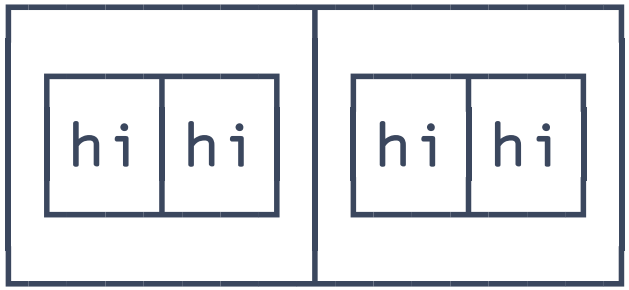
```
      □←t1←'hi'
hi
```


Function Application

```
⍞←t2←2⍴⊂t1←'hi'
```

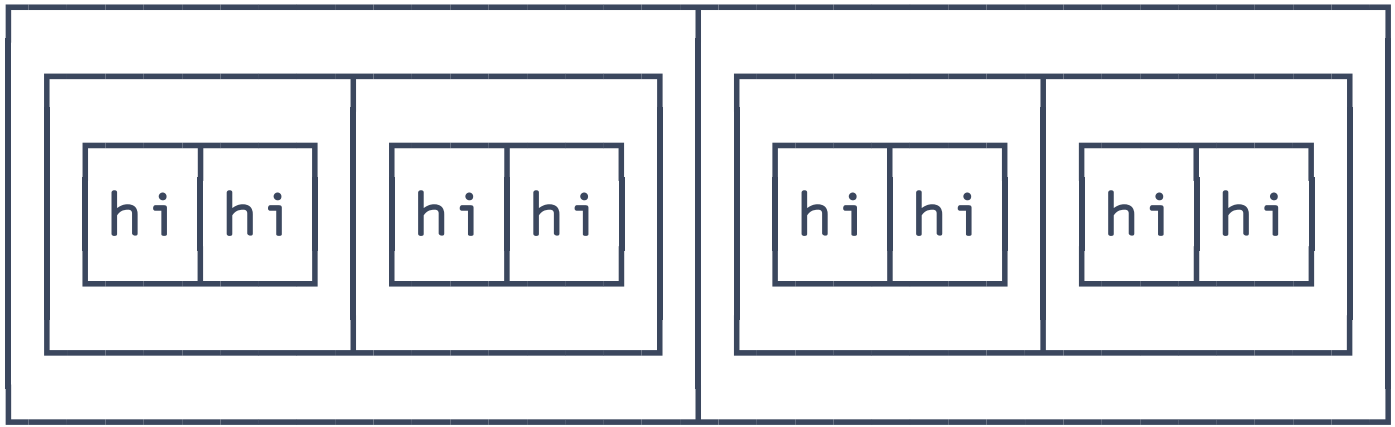
Function Application

```
□←t3←2⍴⊂t2←2⍴⊂t1←'hi'
```



Function Application

```
⍬←t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'
```



Function Application

```
      t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

      '^.'⎕R'\u&'⊢t1
Hi
```


Function Application

```
      t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

      '^.'⌷R'\u&'⊢t2
```

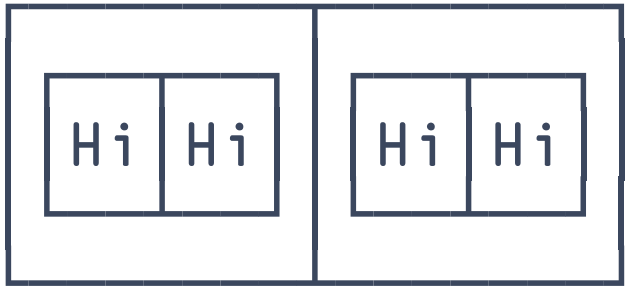
Hi	Hi

Function Application

```
      t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

      '^.'⍞R'\u&'⊢t3
DOMAIN ERROR: Invalid input source
      '^.'⍞R'\u&'⊢t3
          ^
```

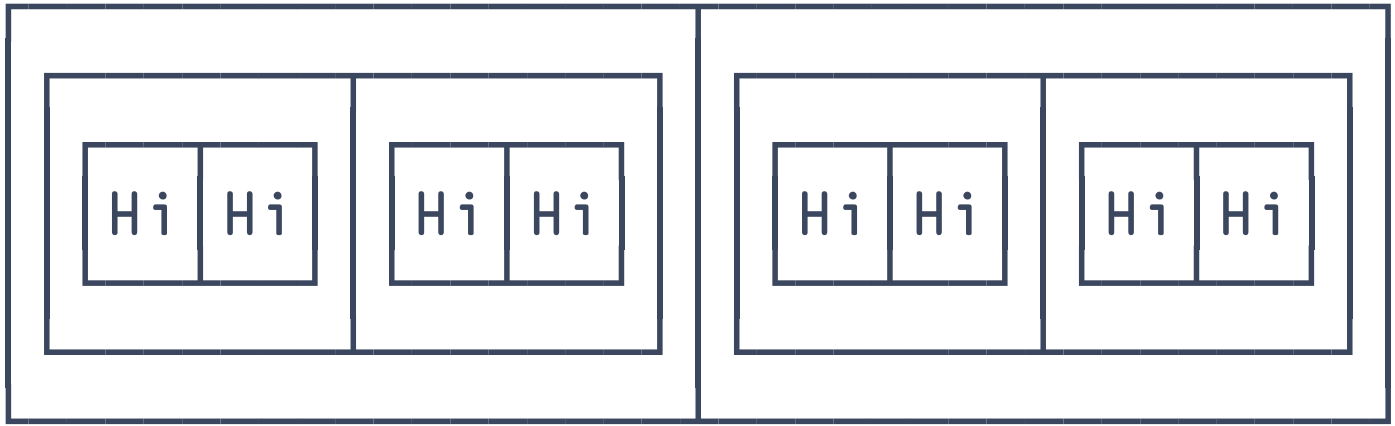

Function Application

```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⍞R'\u&'⍤2⊢t3
```



Function Application

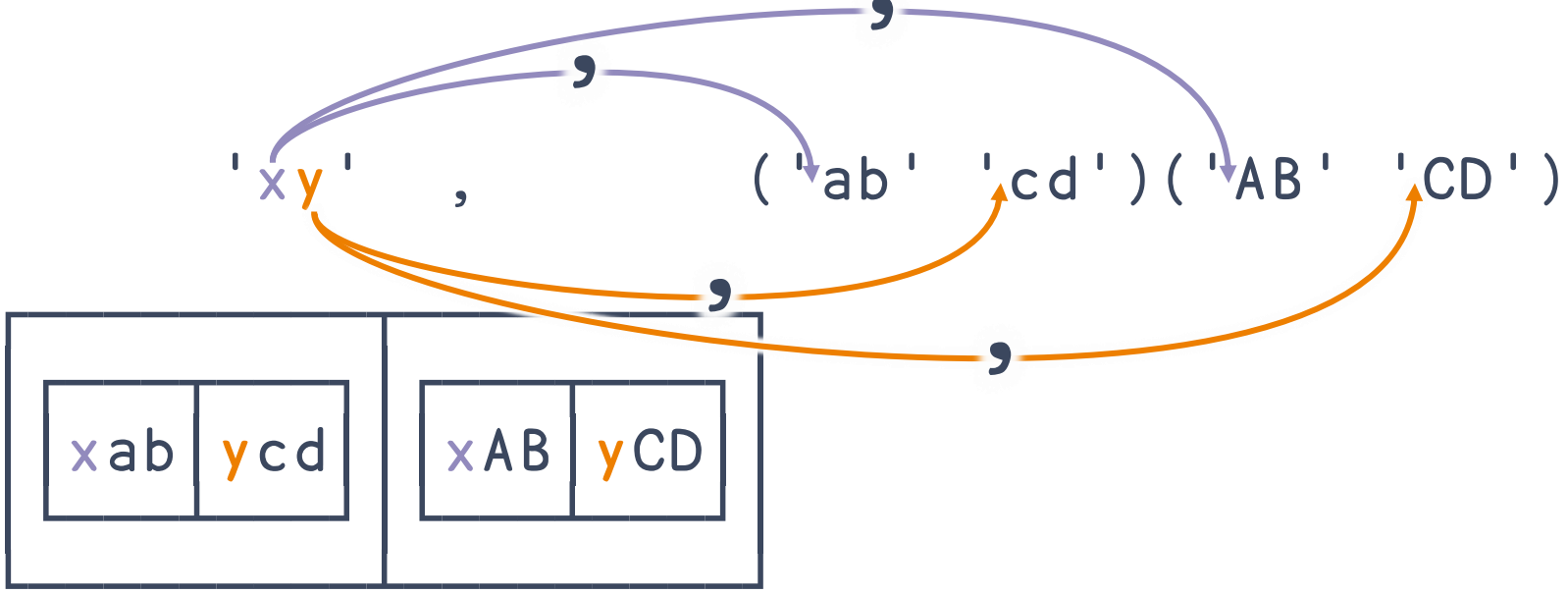
```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⍞R'\u&'⍤2⊢t4
```



Function Application

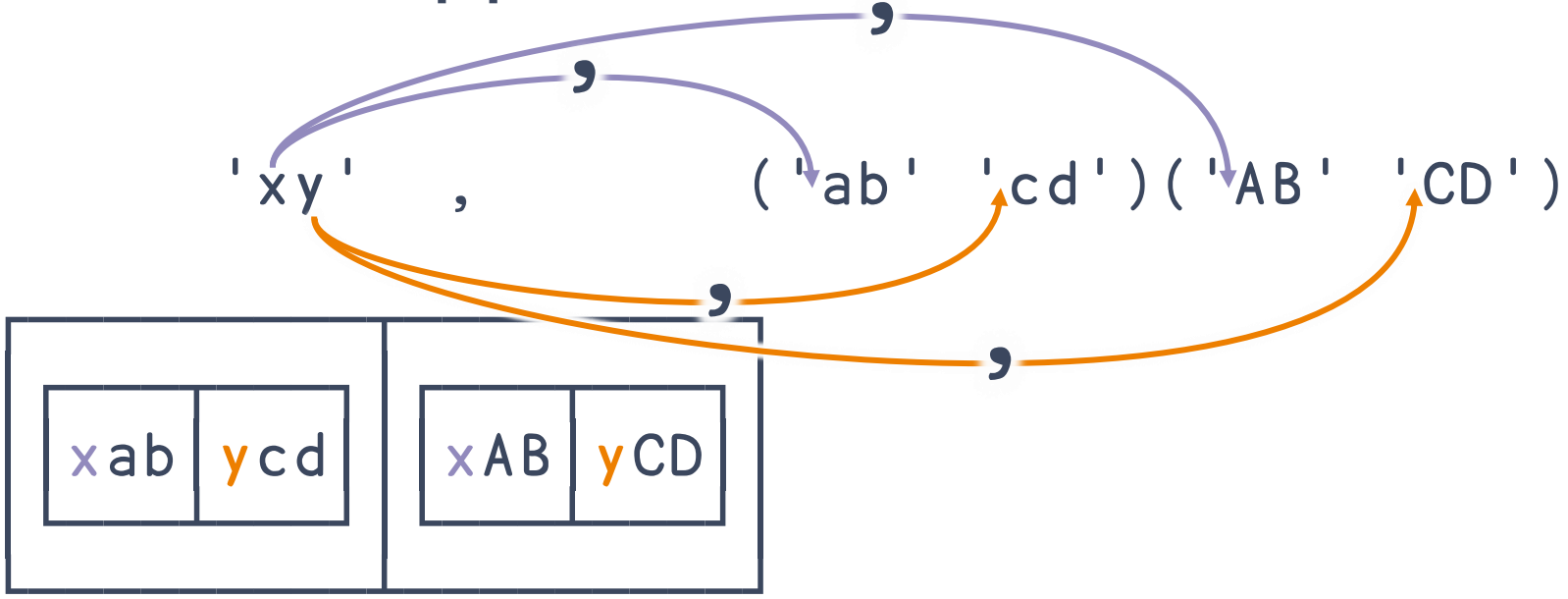
```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⎕R'\u&'⍤2⊢t4
{2≤|≡⍵:∇¨⍵ ⋄ '^.'⎕R'\u&'⊢⍵}
```


Function Application



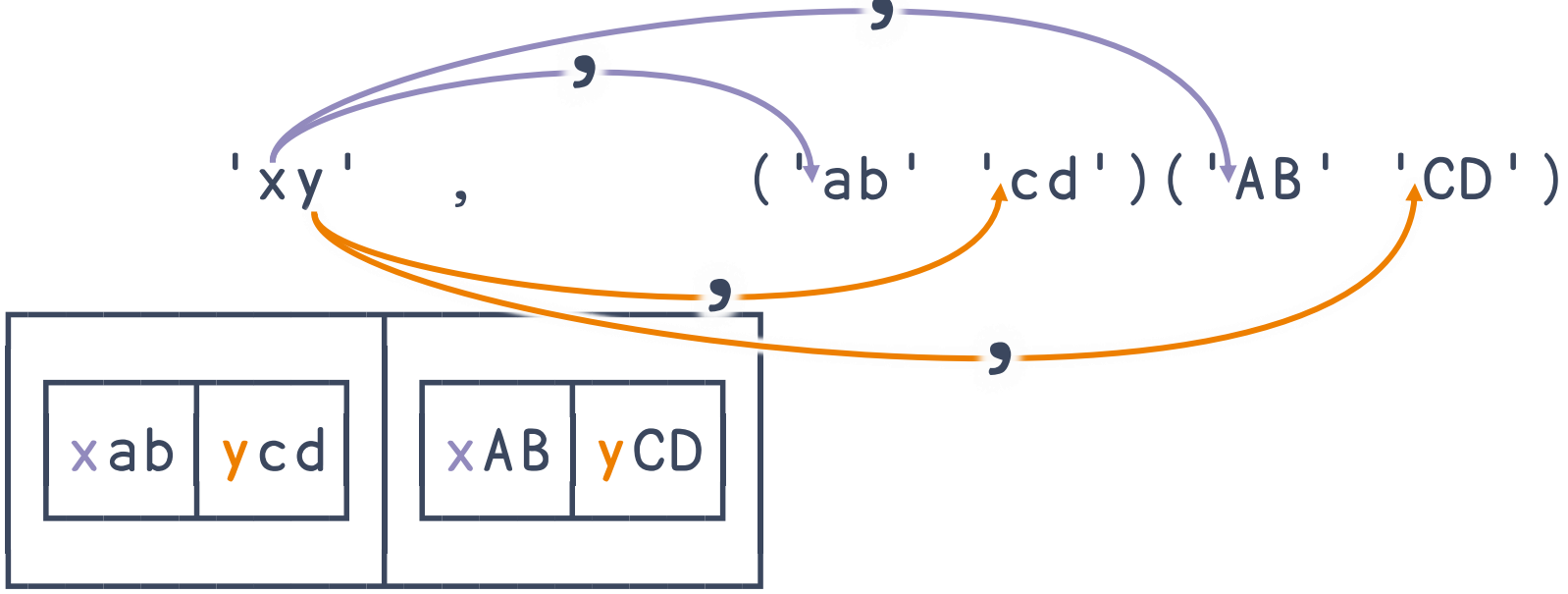

```
(⊂'xy') ,⍤⍤      ('ab' 'cd')('AB' 'CD')
  'xy' (,⍤⍤⍥1 2) ('ab' 'cd')('AB' 'CD')
```


Function Application




```
(⊂'xy') ,⍤⍤      ('ab' 'cd')('AB' 'CD')
  'xy' (,⍤⍤2)     ('ab' 'cd')('AB' 'CD')
```


Function Application




```
(⊂'xy') ,⍤⍤      ('ab' 'cd')('AB' 'CD')
  'xy'  ,⍤⍤⍥2⊢    ('ab' 'cd')('AB' 'CD')
```


Core Language

Data Transformation

Function Application

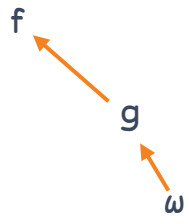
Function Composition

Core Language

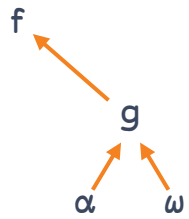
Data Transformation	X × Y	ϕ Y	X [] Y	X ⊒ Y
Function Application	f ≠	f ⁑ g	f ⍥ k	f ⍤ k
Function Composition	f ⍥ g	f ⍤ g	f ∘ g	

Function Composition

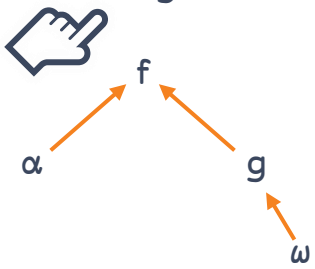
f ⍤ g f ∘ g f ⍥ g



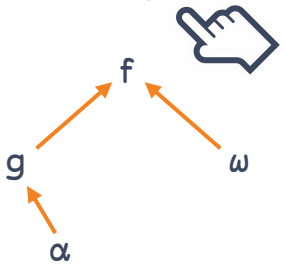
f ⍤ g



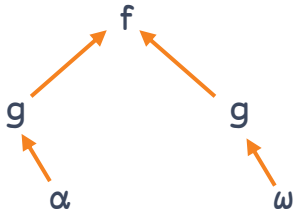
f ∘ g



f ⍨ ∘ g ⍨

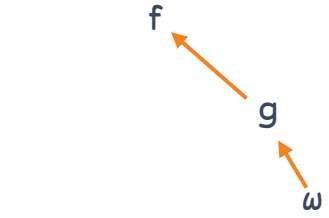


f ⍥ g

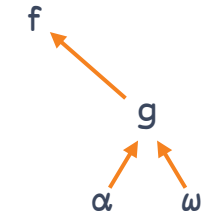


Function Composition

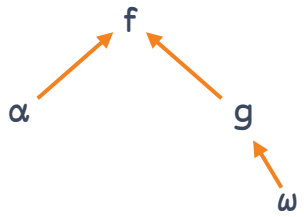
f ⍤ g f ∘ g f ⍥ g



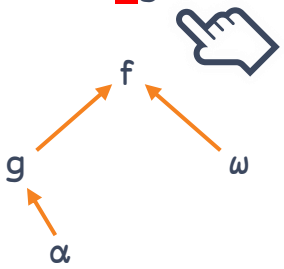
f ⍤ g



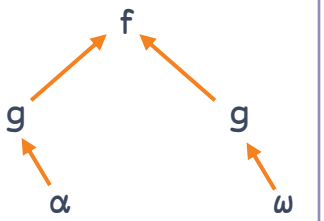
f ∘ g



f ? g



f ⍥ g



Function Composition

Function Composition

Behind

Function Composition

Behind

$$f \circ g$$

$$X \to (f \circ g) \to Y$$

Function Composition

Behind

$$f \circ g$$

$$X \xrightarrow{f} \quad \xrightarrow{g} Y$$

Function Composition

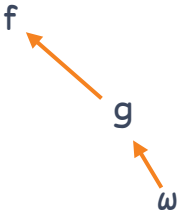
Behind

$$f \circ g$$

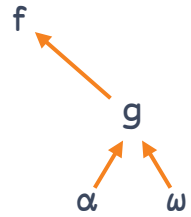
$$(f\ X)\ g\ Y$$

Function Composition

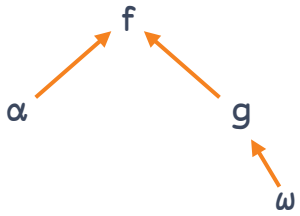
f ⍤⍤ g f ∘ g f ⍥ g



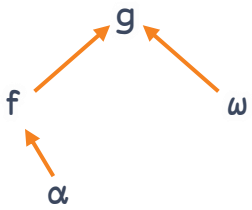
f ⍤⍤ g



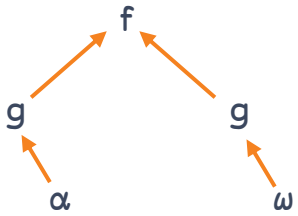
f ∘ g



f ∘ g

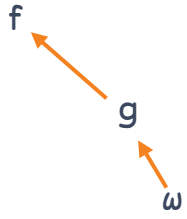


f ⍥ g

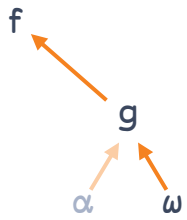


Function Composition

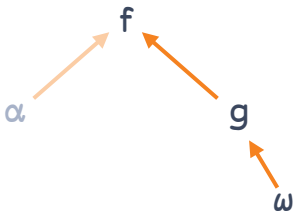
f ⍤ g f ∘ g f ⍥ g



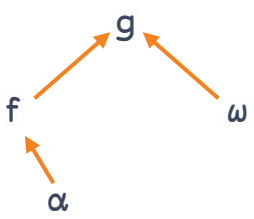
f ⍤ g



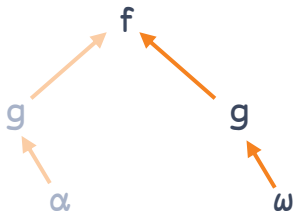
f ∘ g



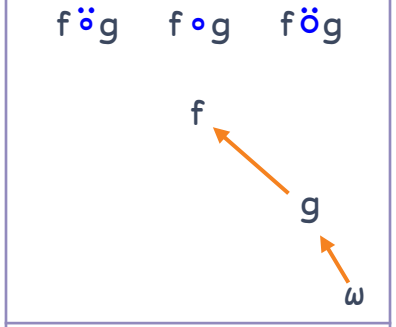
f ∘ g

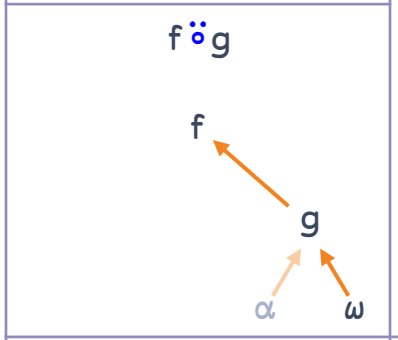


f ⍥ g

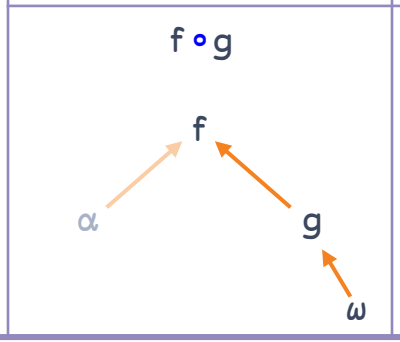


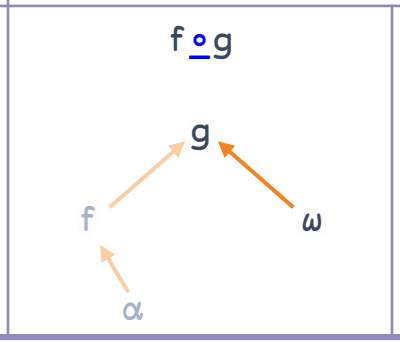
Function Composition

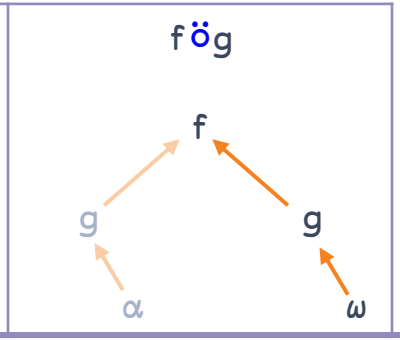




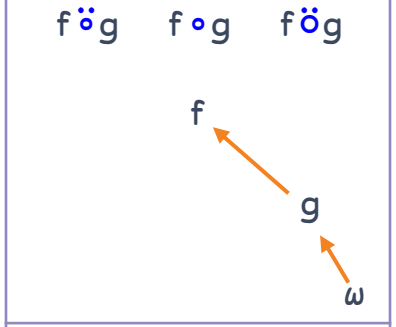
f ∘ g ω ⇔ g ω

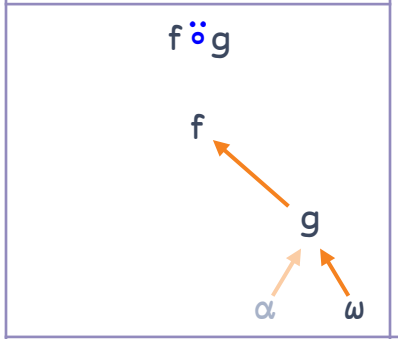




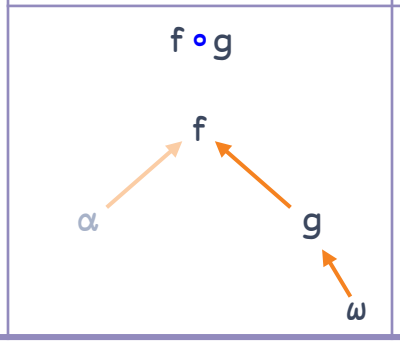


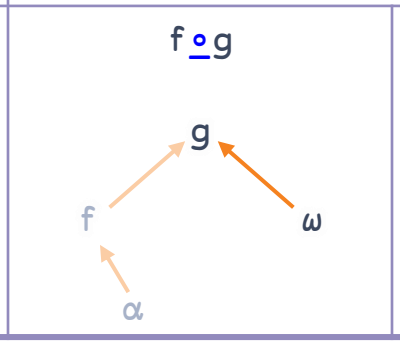
Function Composition

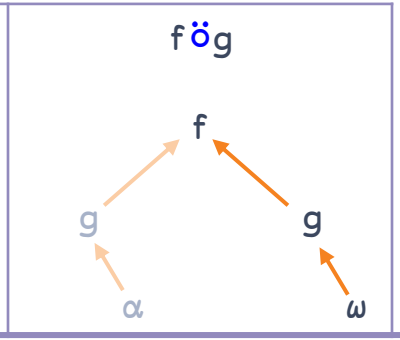




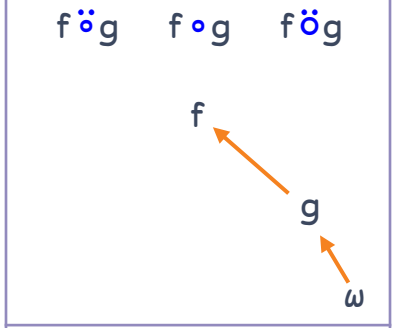
f ∘ g ω ⇔ g ω

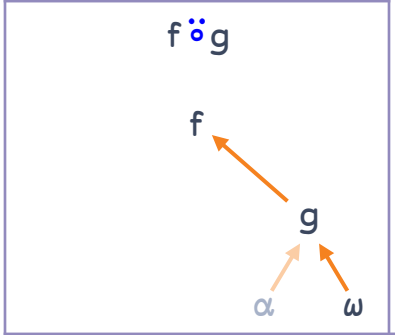


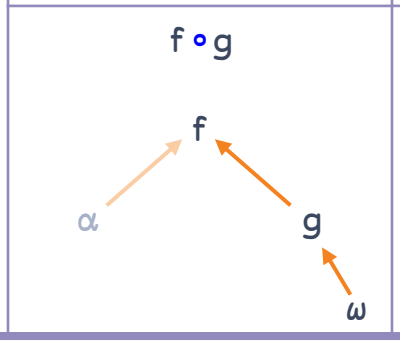


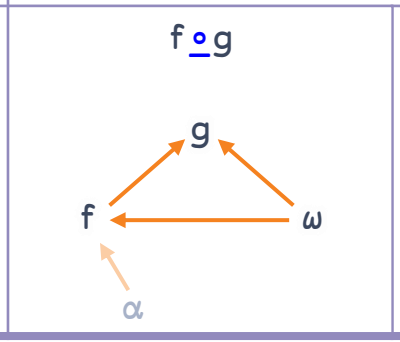


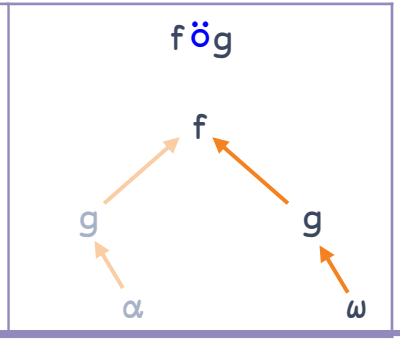
Function Composition











$$f \circ g \ \omega \Leftrightarrow f \circ g \tilde{\ddot{\omega}}$$

Essays/Hook Conjunction?

≺ Essays

New Users
J code search (new)
NuVoc
J in a browser
Guides
System
Showcase
Library
Community
Recent changes
Prototype wiki

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Hook is a 2-train, an isolated sequence of two verbs, introduced in APL by K.E. Iverson and E.E. McDonnell, *Phrasal Forms*, APL89, APL Quote-Quad, Volume 19, Number 4, 1989-08. It is defined as follows:

```
  (g h) y   ↔   y g h y
x (g h) y   ↔   x g h y
```

For example, the monad `(=<.)` is a test for integers and `(+%)/` computes a continued fraction -- `(+%)/20$1` is an approximation of the golden ratio. Hook is based on the S combinator of combinatory logic .

With over 17 years of hindsight, I believe it would have been better to use a conjunction (denoted by `h.`, say) to denote a hook rather than using a 2-train. Everything that can be done with the 2-train `(f g)` can be done with the conjunction `h.`, but `h.` does not require a special parsing rule.

The original motivation for assigning a meaning to a train of length 2 was so that a train of any length (greater than 1) would be interpreted: A train with odd length is a sequence of forks; a train with even length is either a hook (if of length 2) or a hook followed by a sequence of forks (if of length >2). Again with hindsight, the alternatives are:

0. Leave trains of even length uninterpreted -- just signal error.

1. Assign the "at" meaning to it:

```
    (g h) y   ↔  g   h y
  x (g h) y   ↔  g x h y
```

That is, the capped fork meaning. You'd probably still have the capped fork. Compare:

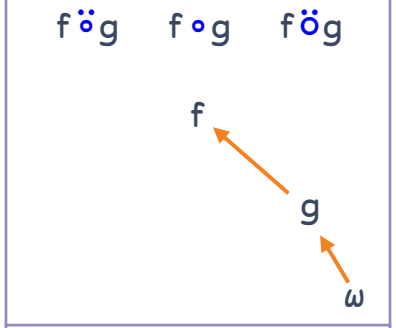
```
  [: f0 [: f1 f2 f3 f4
  (f0 (f1 (f2 f3 f4)))
```

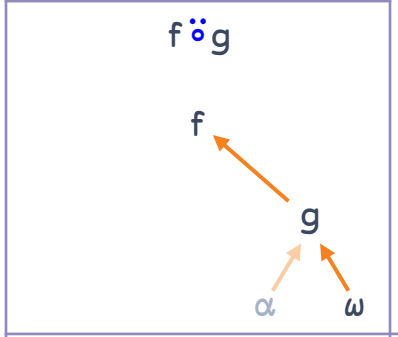
See also

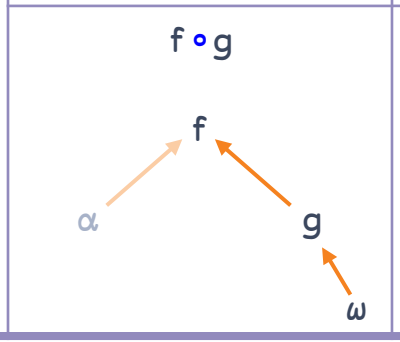
- Trains

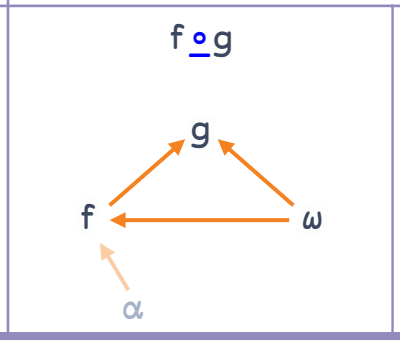
Contributed by Roger Hui.

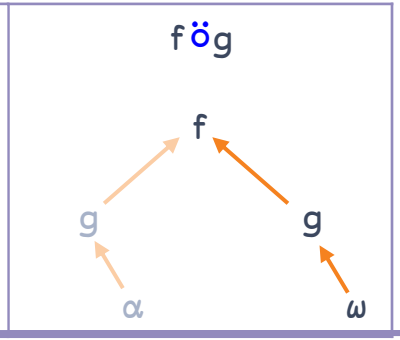
Function Composition







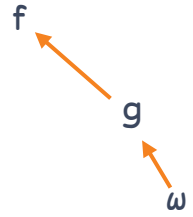




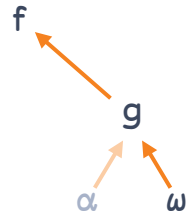
$$f \circ g \ \omega \Leftrightarrow f \circ g \tilde{\odot} \omega$$

Function Composition

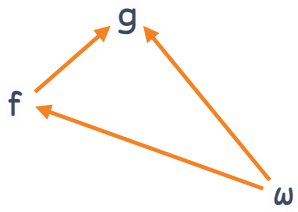
f ⍤⍤ g f ∘ g f ⍥ g



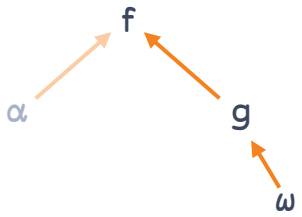
f ⍤⍤ g



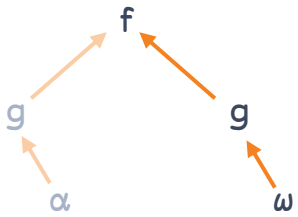
f ⍤ g



f ∘ g



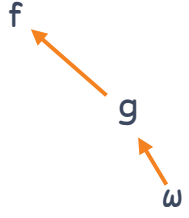
f ⍥ g



Function Composition

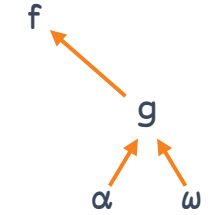
f⍥g f∘g f⍤g

pre/postprocess



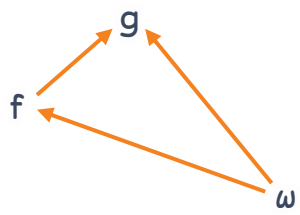
f⍥g

postprocess



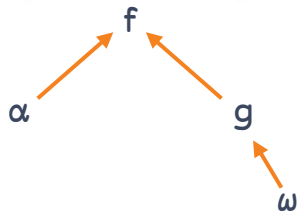
f⍤g

preprocess left



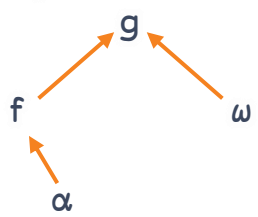
f∘g

preprocess right



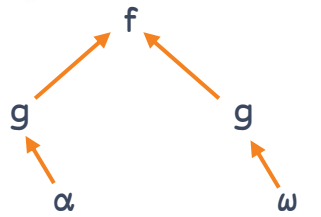
f⍤g

preprocess left



f⍥g

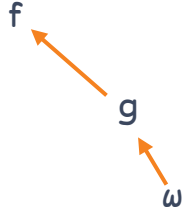
preprocess both



Function Composition

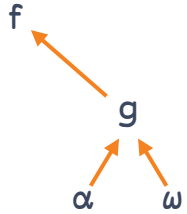
f⍤g f∘g f⍥g

pre/postprocess



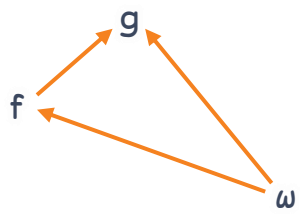
f⍤g

postprocess



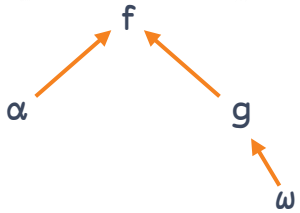
f⍛g

preprocess left



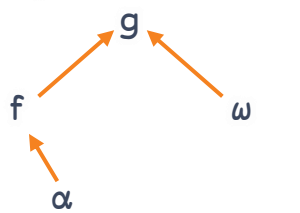
f∘g

preprocess right



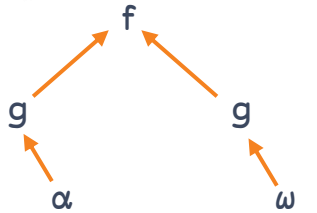
f⍛g

preprocess left



f⍥g

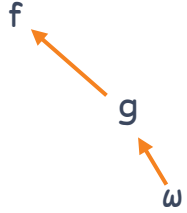
preprocess both



Function Composition

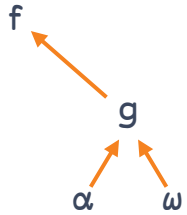
f⍤⍤g f∘g f⍥g

pre/postprocess



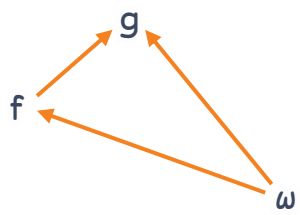
f⍤⍤g

postprocess



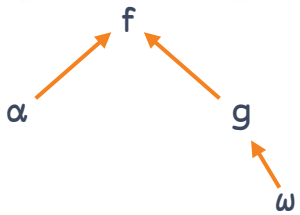
f∘g

preprocess left



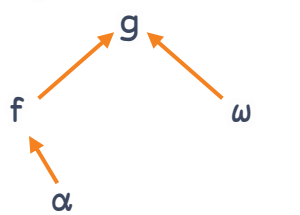
f∘g

preprocess right



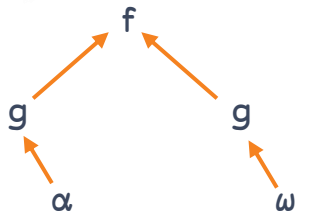
f∘g

preprocess left

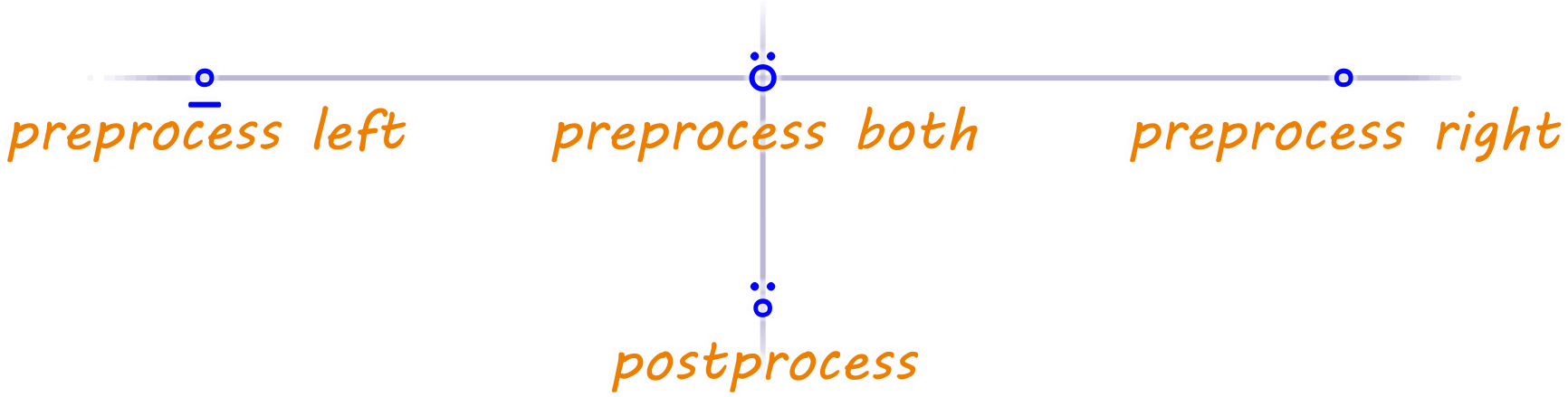


f⍥g

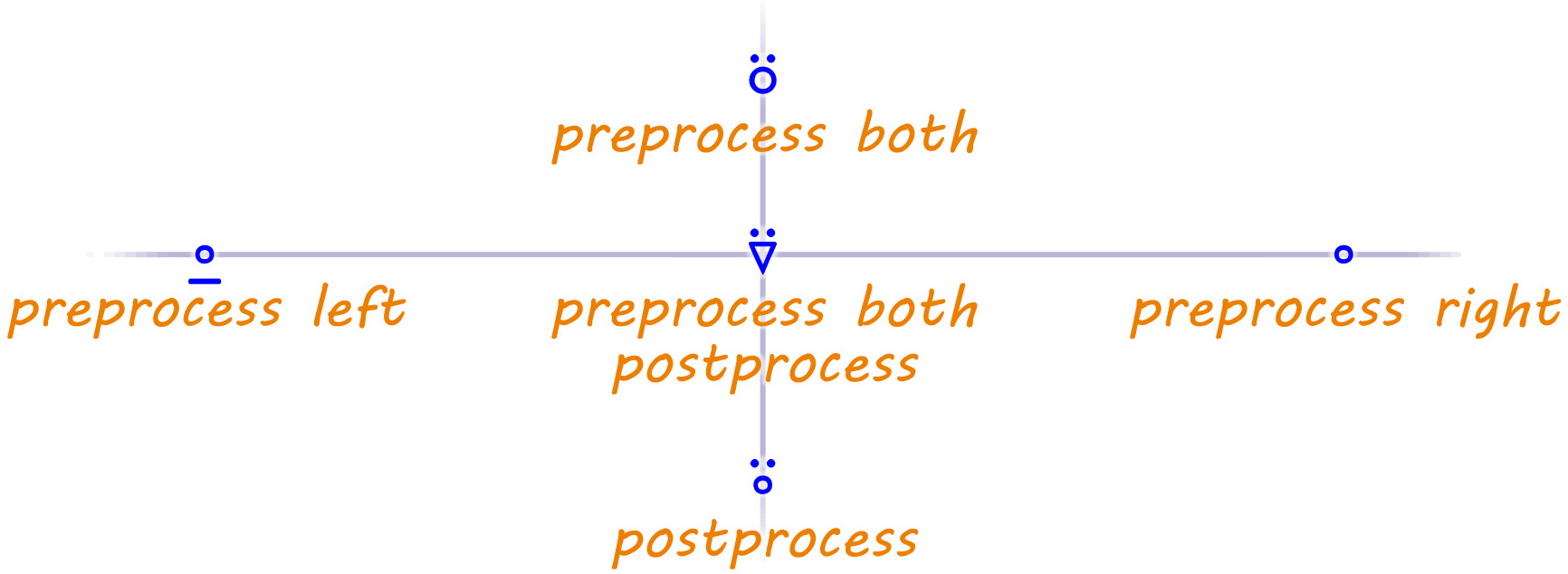
preprocess both



Function Composition



Function Composition



Function Composition

`f⍥g` Behind with `X⍃Y` Select/Permute

- `Sort ← (⍋⌷⊢)`

Function Composition

`f∘g` Behind with `X⊇Y` Select/Permute

- `Sort ← ⍋⌷⊢`

Function Composition

`f⍥g` Behind with `X⍃Y` Select/Permute

```
  Sort  ← ⍋⍥⍃
● Sorts ← ⍃⍨∘⍋⍨      ⍝ "sort Y by X"
```


Function Composition

`f⍥g` Behind with `X⍃Y` Select/Permute

```
  Sort   ← ⍋⍥⍃
● Sorts  ← ⍋⍥⍃      ⍝ "sort Y by X"
```

Function Composition

`f⍥g` Behind `X⍸Y` Select/Permute

```
Sort   ← ⍋⍥⍸
Sorts  ← ⍋⍥⍸      ⍝ "sort Y by X"
Shuffle ← (?⍨⍥≠⍸⊢)
```


Function Composition

`f⊸g` Behind with `X⊇Y` Select/Permute

```
Sort    ← ⍋⊸⊇
Sorts   ← ⍋⊸⊇          ⍝ "sort Y by X"
Shuffle ← ?˜⊸≠⊸⊇
```


Function Composition

f ∘ g Behind

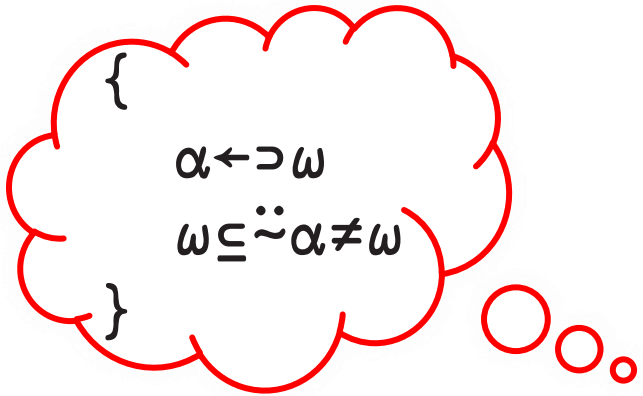
- SameAsFirst ← ↄ∘=
- HasDuplicates ← ∪∘≡
- Palindrome ← ⌽∘≡
- IsPermutation ← ⍋⍤⍋∘≡

Function Composition

f⍥g Behind

```
Integer   ← ⌊⍥=
Split     ← (⊃⍥≠⊆⊢)
Scale     ← ⌈/⍥(÷⍨)
Deviation ← (+⌿÷≠)⍥(-⍨)
```

```
'/' (≠⊆⊢)  'hi/how/goes'
   (⊃⍥≠⊆⊢)'/hi/how/goes'
'/'(⊃⍥≠⊆⊢)'/hi/how/goes'
```



Function Composition

f ∘ g Behind

```
      Filters ← ∘≠
      >∘5 Filters 2 7 1 8 2 8
7 8 8
      ⌽∘≡¨Filters 'racecar' 'racer' 'toot'
 racecar  toot
```


Function Composition

`f⍥g` Behind

```
      >⍥5 ⍥≠ 2 7 1 8 2 8
7 8 8
      ⍞⍥≡¨⍥≠ 'racecar' 'racer' 'toot'
 racecar  toot
```


Function Composition

f⍥g Behind

- `Whence ← ⍳⍥∈` `⍝ {(⍳⍺)∈⍵}`
- `InPoly ← ⍪⍥⊥` `⍝ {(⍪⍺)⊥⍵}`
- `Shapes ← ⍴⍥⍴` `⍝ {(⍴⍺)⍴⍵}`
- `ToFile ← ⊂⍥⎕NPUT` `⍝ {(⊂⍺)⎕NPUT ⍵}`

Function Composition

f⍥g Behind

- `FCat ← ⫢⍥,`

```
RIndex ← ⍳⍥⊖        ⍝ {(⊖⍺)⍳⍵}
RDrop  ← ↓⍥-        ⍝ {(-⍺)↓⍵}
RndSfx ← ↑⍥(-?⍨)    ⍝ {(-?⍺)↑⍵}
```

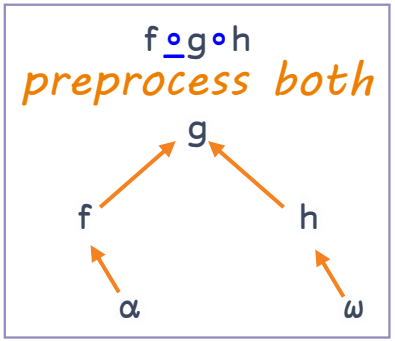
APL\360: `'abc';42;'def'`
Dyalog: `'abc',(⫢42),'def'`
20.0: `'abc',42⫢⍥,'def'`

Function Composition

`f ⊙ g` Behind

Split-compose `X f ⊙ g ⊙ h Y`

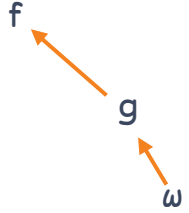
Function Composition



Function Composition

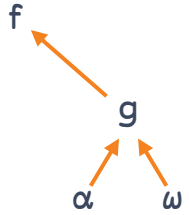
f⍤g f∘g f⍥g

pre/postprocess



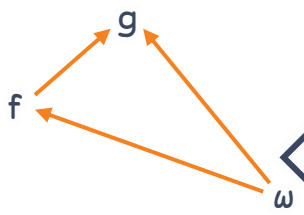
f⍤g

postprocess



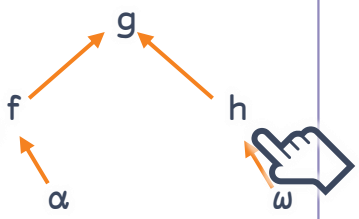
f∘g

preprocess left



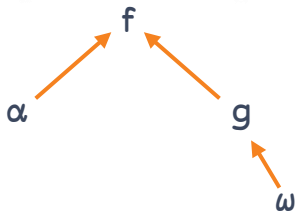
f∘g∘h

preprocess both



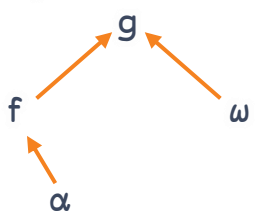
f∘g

preprocess right



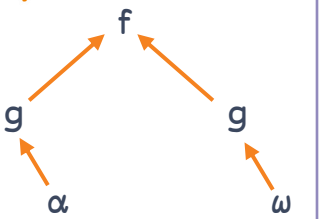
f∘g

preprocess left



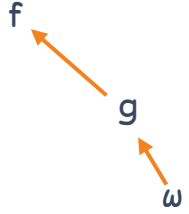
f⍥g

preprocess both

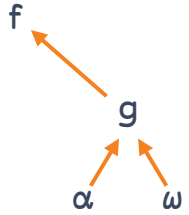


Function Composition

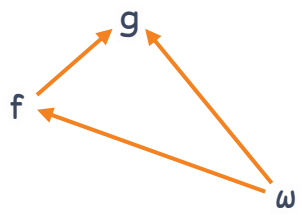
f⍥g f⍤g f⍥g
pre/postprocess



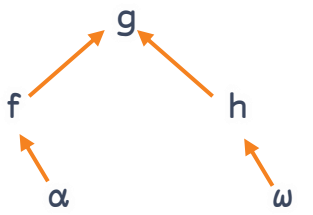
f⍥g
postprocess



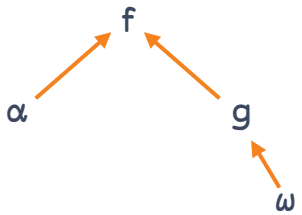
f⍤g
preprocess left



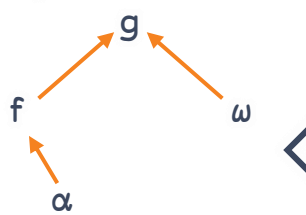
f⍤g⍤h
preprocess both



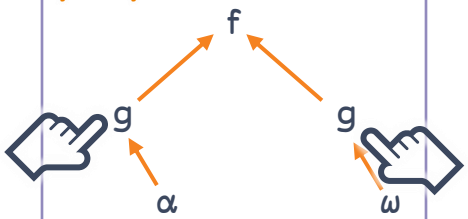
f⍤g
preprocess right



f⍤g
preprocess left

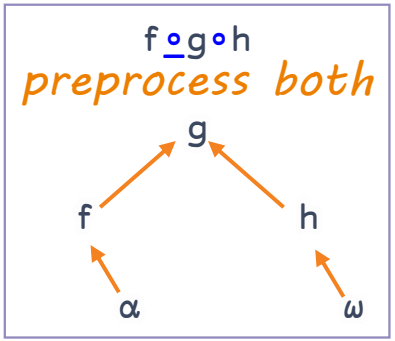


f⍥g
preprocess both



α g⍤f⍤g ω
⇔
α f⍥g ω

Function Composition



Function Composition

`f⍥g` Behind

Split-compose `X f⍥g∘h Y`

Pre-18.0: `g⍨∘f⍨∘h`
18.0: `(f⍤⊣ g h⍤⊢)`
20.0: `f⍥g∘h`

Function Composition

`f⍥g` Behind

Split-compose `X f⍥g∘h Y`

```
Pre-18.0: ((f⊣)g(h⊢))
18.0:     (f⍤⊣ g h⍤⊢)
20.0:        f⍥g∘h
```

Function Composition

`f⍥g` Behind

Split-compose `X f⍥g∘h Y`

Hybrid mitigation `2∘| ⍥/`

Pre-18.0: `(2∘|{⍺/⍵}⊢)`
18.0: `(2∘|⊢⍤⍥/⊢)`
20.0: `2∘| ⍥/`

Language Enhancements

Data Transformation

Function Application

Function Composition

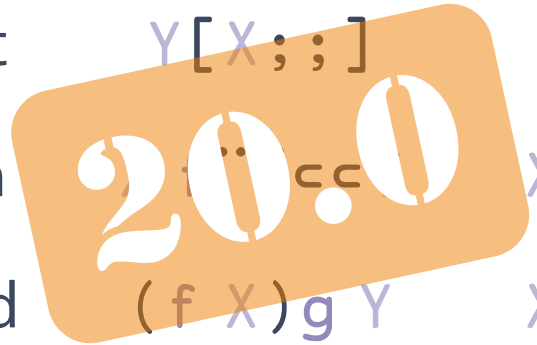
$f \circ g$

Language Enhancements

Data Transformation	X × Y	ϕ Y	X [] Y	X ⩲ Y
Function Application	f ≠	f ⁑ g	f ⊙̈ k	f ⊙̈ k
Function Composition	f ⊙̈ g	f ⊙̈ g	f ∘ g	f ⊜ g

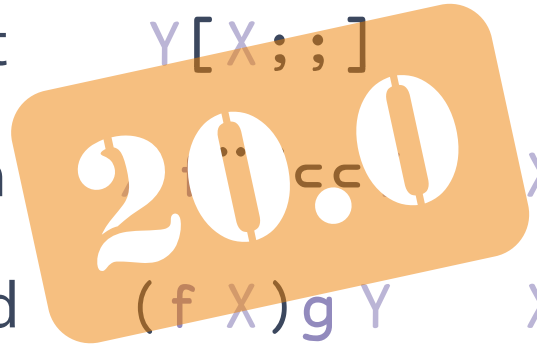
Language Enhancements

Data Transformation	Select	`Y[X;;]`	`X⊒Y`
Function Application	Depth	`f⍤⊂⊂`	`X f⍤k Y`
Function Composition	Behind	`(f X)g Y`	`X f⍥g Y`



Language Enhancements

Data Transformation	Select	`Y[X;;]`	`X⊇Y`
Function Application	Depth	`f⍤⊂⊂`	`X f⍤⊂⊂ Y`
Function Composition	Behind	`(f X)g Y`	`X f⍥g Y`
Axis Manipulation			



Axis Manipulation

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \otimes \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}$$

`A×⊂B`

5 6 7 8	10 12 14 16
15 18 21 24	20 24 28 32

`⍪≠A×⊂B`

5 6	10 12
7 8	14 16
15 18	20 24
21 24	28 32

`,/⍪≠A×⊂B`

5	6	10	12
7	8	14	16
15	18	20	24
21	24	28	32

`⊃,/⍪≠A×⊂B`

5 6 10 12
7 8 14 16
15 18 20 24
21 24 28 32

Axis Manipulation

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

A×⊂B

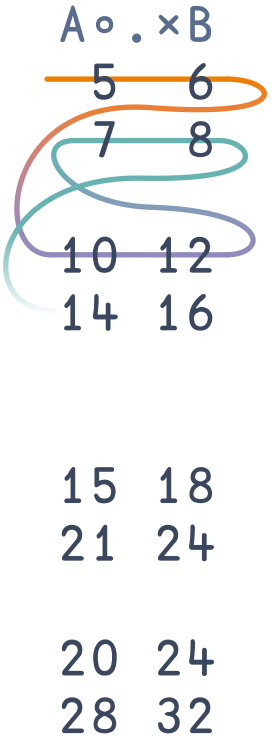
5 6 7 8	10 12 14 16
15 18 21 24	20 24 28 32

Axis Manipulation

```
A ∘.× B
 5  6
 7  8

10 12
14 16


15 18
21 24

20 24
28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation



$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
1 2 3 4⍉A∘.×B
```

```
 5  6
 7  8

10 12
14 16


15 18
21 24

20 24
28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
1 3 2 4⍉A∘.×B
 5  6
10 12

 7  8
14 16


15 18
20 24

21 24
28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
,[1 2] 1 3 2 4⍉A∘.×B
 5  6
10 12

 7  8
14 16

15 18
20 24

21 24
28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
      ,[2 3] ,[1 2] 1 3 2 4⍉A∘.×B
 5  6 10 12
 7  8 14 16
15 18 20 24
21 24 28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
,[1 2]⍤¯1 ,[1 2] 1 3 2 4⍉A∘.×B
 5  6 10 12
 7  8 14 16
15 18 20 24
21 24 28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
,[1 2]⍤2 ,[1 2] 1 3 2 4⍉A∘.×B
 5  6 10 12
 7  8 14 16
15 18 20 24
21 24 28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

```
?⍤2 ? 1 3 2 4⍉A∘.×B
 5  6 10 12
 7  8 14 16
15 18 20 24
21 24 28 32
```

$$\begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Axis Manipulation

Axis Manipulation

Demote

Axis Manipulation

Promote

^Y

Axis Manipulation

```
'Hallo'
```

Promote

```
^Y
```

Axis Manipulation

```
      ⍴'Hallo'
5
```

Promote

```
      ⍉Y
```

Axis Manipulation

```
    ≠⍴'Hallo'
1
```

Promote

```
^⍤Y
```

Axis Manipulation

```
      ≠p^'Hallo'
2
```

Promote

Axis Manipulation

```
      ≠p^^'Hallo'
3
```

Promote

Axis Manipulation

```
      ^'Hallo'
```

```
⎕ML<1: ⊃
```

```
⌽⍪'Hallo'
↑,⊂'Hallo'
1 5⍴'Hallo'
,[÷2]'Hallo'
,[⎕IO-÷2]'Hallo'
```

```
  'Hallo'⍪⍤⊂^'Welt!'
Hallo
Welt!
```

```
↑'Hallo' 'Welt!'
```

```
⎕ML<1: ⊃
```

Axis Manipulation

```
      ⍋'Hallo'
                                  ⍒⍪'Hallo'
                                  ↑,⊂'Hallo'
                                  1 5⍴'Hallo'
∨     ,[⍳2]                 ∧     ,[÷2]'Hallo'
                ,[⍤f⍤2]           ,[⎕IO-÷2]'Hallo'

      'Hallo'⍪⍤⍋∧⍤'Welt!'       ↑'Hallo' 'Welt!'
Hallo
Welt!
```

Axis Manipulation

```
      ∧'Hallo'                          ⍉⍪'Hallo'
                                        ↑,⊂'Hallo'
                                        1 5⍴'Hallo'
∨    ,[⍳2]                          ∧   ,[÷2]'Hallo'
              ,[⍳÷2]                    ,[⎕IO-÷2]'Hallo'

     'Hallo'⍪⍥∧'Welt!'                  ↑'Hallo' 'Welt!'
Hallo
Welt!
```

Axis Manipulation

```
      ∧'Hallo'                  ⍉⍪'Hallo'
                                ↑,⊂'Hallo'
                                1 5⍴'Hallo'
∨    ,[⍳2]                  ∧   ,[÷2]'Hallo'
              ,[⍳÷2]            ,[⎕IO-÷2]'Hallo'

    'Hallo'⍪⍤∧'Welt!'           ↑'Hallo' 'Welt!'
Hallo
Welt!
```

Axis Manipulation

```
      (∨\b)∧⍀∨\⍀b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
      ↑
```


Axis Manipulation

```
      (⍉\b)∧⌽⍉\⌽b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
      (⍉\b)∧⌽⍉\⌽b←0 1 0 0 1 1 0 0
```


Axis Manipulation

```
        (∨\b)∧⍀∨\⍀b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b (∨\b)∧⍀∨\⍀b←0 1 0 0 1 1 0 0
```


Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ ⍝(∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
```


Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ¯⍤⊖∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        ⍝(∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
```


Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ¯⍤⍥∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b │(∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
```


Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ¯⍤⍥∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ( |(∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
```

Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ( (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
```

Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ( (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0 )
```


Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ( (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0 )
```

Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        ↑ b ( (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0 )
```

Axis Manipulation

```
        (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        b ⍪⍤∧ (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
        ↑ b ( (∨\b)∧⍱∨\⍱b←0 1 0 0 1 1 0 0 )
0 1 0 0 1 1 0 0
0 1 1 1 1 1 0 0
          |
```

Language Enhancements

Data Transformation

Function Application

Function Composition

Axis Manipulation

Language Enhancements

Data Transformation	Select	`Y[X;;]`	`X⊇Y`
Function Application	Depth	`X f⁼⁼⁼⁼⊂⊂Y`	`X f⍥k Y`
Function Composition	Behind	`(f X)g Y`	`X f⍥g Y`
Axis Manipulation	De/Promote	`,[⍪2]Y`	`⌄Y ⌃Y`