

DYALOG

Elsinore 2019



A Decade of APL Extensions Grouping and Processing Text

Nicolas Delcros

Partitioning: \Subset and \subseteq

- Partition an array (along an axis)
- into nested arrays (depth increases by 1)
- preserving order of appearance
- ω gives the array (what to cut)
- α gives the partition (where to cut)
- \Subset is “Partitioned Enclose” and \subseteq is “Partition”

'ABCDEF' $\rightarrow\rightarrow$ 'AB' 'CDEF'



$$\alpha \subset \omega$$

Partitioned Enclose

- α is boolean
 ω is cut at 1's
- May only discard leading items of ω if α has leading 0's
- High-rank:
Return vector of arrays

$$\alpha \subseteq \omega$$

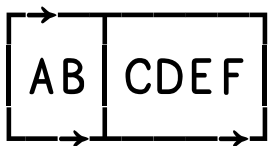
Partition

- α is non-negative integer
 ω is cut at increases of $\alpha > 0$
- Can discard arbitrary items of ω with $\alpha = 0$
- High-rank:
Return array of vectors

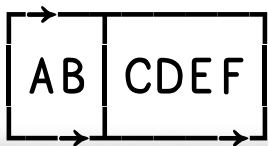


Left argument

- $\alpha \subset \omega$: boolean α , cuts at 1's
`]disp 1 0 1 0 0 0 ⊂ 'ABCDEF'`



- $\alpha \subseteq \omega$: integer α , cut at increases of positive α
`]disp 1 1 3 3 3 1 ⊆ 'ABCDEF'`



Left argument

- Exercise: Convert left argument of \subset into left argument of \subseteq

- Behaviour:

$\{...\}$ 1 0 1 0 0 0

1 1 2 2 2 2

- Solution:

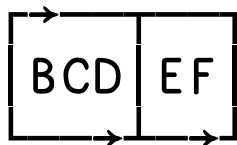
$\alpha \subset \omega \leftrightarrow (+ \backslash \alpha) \subseteq \omega$ α for any boolean α



Left argument: discarding items

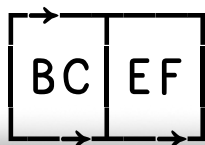
- $\alpha \subset \omega$: all items of ω end up in the result excepted 0's before the first 1 in α

]disp 0 1 0 0 1 0 \subset 'ABCDEF'



- $\alpha \subseteq \omega$: any item of ω can be discarded with $\alpha = 0$

]disp 0 1 1 0 1 1 \subseteq 'ABCDEF'



Left argument: discarding items

- Exercise: prevent any item of ω to disappear from $\alpha \subset \omega$
- Behaviour:

			{...}	0	0	0	1	0	1	0
1	0	0	1	0	1	0	0	0		

- Solutions:

 $(1, 1 \downarrow \alpha) \subset \omega$

A two “heavy” operations

 $(1 @ \square IO \vdash \alpha) \subset \omega$

A better (in-place)



Left argument

- Exercise: convert left argument of \subseteq into left argument of \subset
- Solution:
impossible because \subset cannot discard arbitrary items of ω



Left argument

- Exercise: convert positive left argument of \subseteq into left argument of \subset

- Solution:

$$\alpha \subseteq \omega \leftrightarrow (1, 2 < / \alpha) \subset \omega \quad \text{if for any } \alpha > 0$$

$$\alpha \subseteq \omega \leftrightarrow (2 < / 0, \alpha) \subset \omega \quad \text{if allow leading zeros}$$



Left argument

- Exercise: rewrite \subseteq with \subset
- Solution:
 1. Discard items where $\alpha=0$
 2. Convert positive left argument of \subseteq into left argument of \subset
$$\{m \leftarrow \alpha > 0 \quad \diamond \quad (m/2 < / 0, \alpha) \subset (m/\omega)\}$$



Converting between \subset and \subseteq

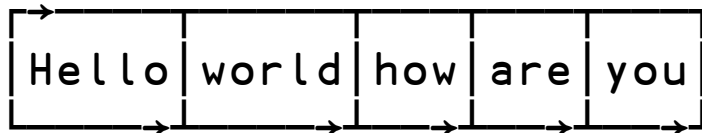
- $\{\alpha \subset \omega\} \leftrightarrow \{(+ \setminus \alpha) \subseteq \omega\}$
- $\{\alpha \subseteq \omega\} \leftrightarrow \{m \leftarrow \alpha > 0 \ \diamond \ (m/2 < / 0, \alpha) \subset (m/\omega)\}$



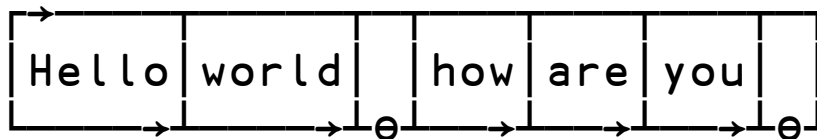
Exercise: cut string at separators

- Exercise: cut string ω at separators α with \subset or \subseteq
- Behaviour:

```
]disp ' .,!? ' {...} 'Hello world, how are you?'
```



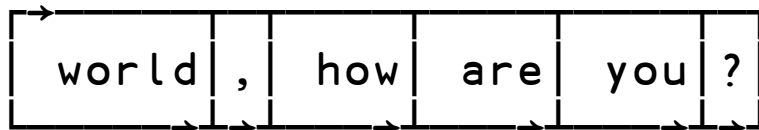
```
]disp ' .,!? ' {...} 'Hello world, how are you?'
```



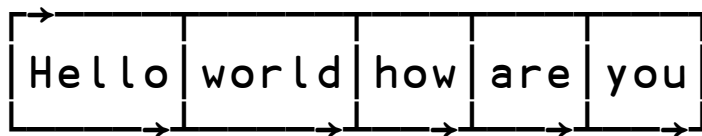
Exercise: cut string at separators with \subseteq

- Solution:

```
]disp ' .,!? ' {(+\omega\alpha)\subseteq\omega} 'Hello world, how are you?'
```



```
]disp ' .,!? ' {(\sim\omega\alpha)\subseteq\omega} 'Hello world, how are you?'
```



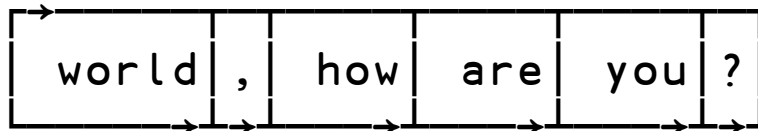
- Problem: inability to cut empty strings (easily)



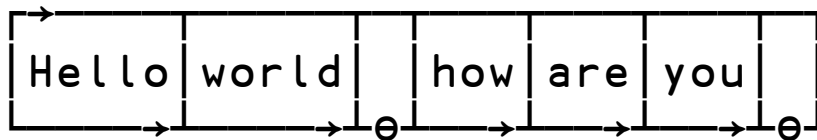
Exercise: cut string at separators with \complement

- Solution:

```
]disp ' .,!? ' {(w $\in$  $\alpha$ ) $\complement$ w} 'Hello world, how are you?'
```



```
]disp ' .,!? ' {l $\leftarrow$ ( $\supset$  $\alpha$ ),w  $\diamond$  1 $\downarrow$ ''(l $\in$  $\alpha$ ) $\complement$ l} 'Hello world, how are you?'
```

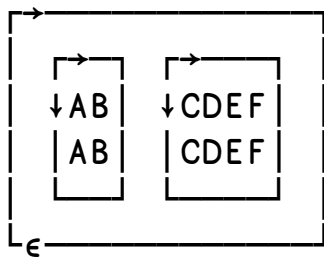


- Problem: catenation and $1\downarrow''$

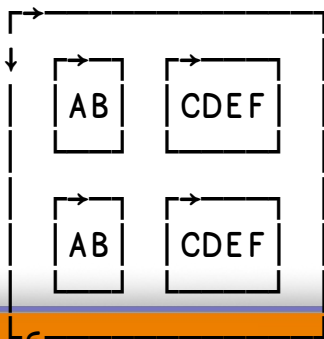


High-rank arrays

```
]display 1 0 1 0 0 0 ⍋[2] 2 6ρ'ABCDEF'
```



```
]display 1 1 2 2 2 2 ⍋[2] 2 6ρ'ABCDEF'
```



High-rank arrays

- $\alpha \subset \omega$: result is vector, items have rank of ω
- $\alpha \subseteq \omega$: result has rank of ω , items are vectors
- \subset performs better
(fewer nested items for same amount of data)



$$\alpha \subset \omega$$

Partitioned Enclose

- α is boolean
 ω is cut at 1's
- May only discard leading items of ω if α has leading 0's
- High-rank:
Return vector of arrays

$$\alpha \subseteq \omega$$

Partition

- α is non-negative integer
 ω is cut at increases of $\alpha > 0$
- Can discard arbitrary items of ω with $\alpha = 0$
- High-rank:
Return array of vectors



DYALOG

Elsinore 2019



A Decade of APL Extensions – Grouping and Processing Text

Richard Smith

Nic Delcros, Marshall Lochbaum, Richard Park

Search and Replace operators

- Search within and update (replace) text
- Full regular expression handling
- Multiple patterns, function callbacks (unique functionality)



Replace operator

search_pattern □R replace_pattern

search_pattern(s) □R replace_pattern(s)

search_pattern □R callback_fn

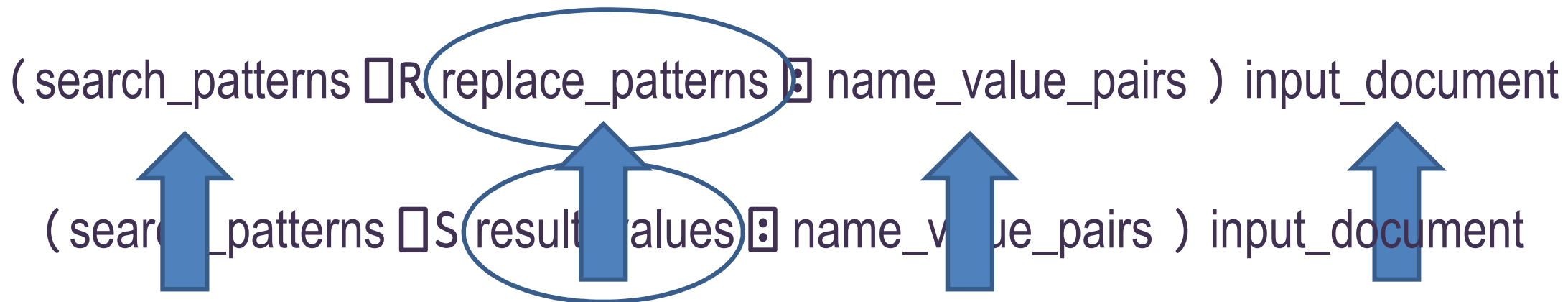
(... □R ...) input_text

output_file (... □R ...) input_file

... (... □R ... ⌈ options) ...

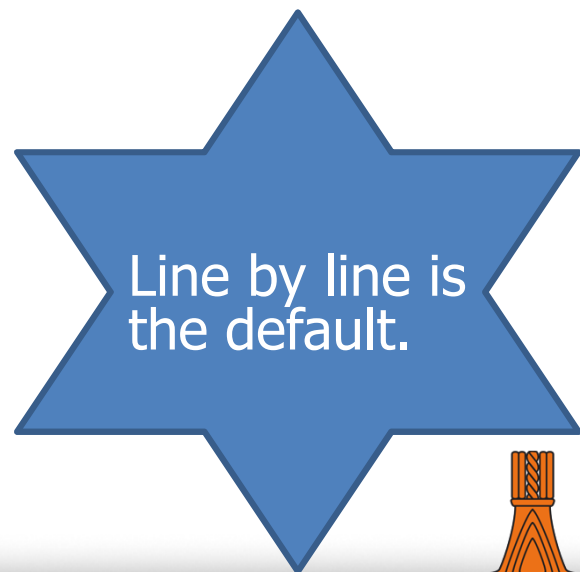


Search and replace operators



Input document

- Represents a single text document as:
 - A vector of character vectors (lines), or
 - A single character vector with embedded line endings
- Character vectors in a vector are *not* independent
- In either case, the document may be processed line-by-line or in its entirety



Search pattern

- 'abc'
- 'a.c'
- 'a.+c'
- 'a(.+)c\1'
- '\S+'
- '\ba.c\b'
- '^ [0-9]*\$'



Anchors



Replace pattern

- ✧ 'x'
- ✧ '-\0-'
- ✧ '\1'
- ✧ '\l0'
- ✧ '&'
- ✧ '%'



Variant options

- ✧ 'IC' 1
- ✧ 1
- ✧ 'Mode' 'D'



What happens here?

```
t←'good day to you'
```

```
('(\w)(\w*)' ⌈R '\u1\l2') t
```

Good Day To You



What happens here?

```
t←'good day to you'
```

```
('DAY' '(\w)(\w*)' ⌈R 'NIGHT' '\u1\l2' ⌈: 1) t
```

```
Good NIGHT To You
```



What happens here?

```
t←'good day to you'
```

```
( '(\w)(\w*)' 'DAY' ⍲R '\u1\u12' 'NIGHT' ⍲ 1) t
```

```
Good Day To You
```



Exercise: an English to Doglish translator

```
t ← 'OK, Ginger! I've had it! You stay out of the  
garbage! Understand, Ginger? Stay out of the garbage, or else!'
```

```
( ... ⍋R ... ) t
```

```
blah GINGER blah blah blah blah blah blah blah blah  
GINGER blah blah blah blah blah blah blah...
```



Exercise: an English to Doglish translator

```
t ← 'OK, Ginger! I've had it! You stay out of the  
garbage! Understand, Ginger? Stay out of the garbage, or else!'
```

```
('ginger\pP*' '\S+' '$' ⌈R 'GINGER' 'blah' '...' ⌋ 1) t
```

```
blah GINGER blah blah blah blah blah blah blah blah  
GINGER blah blah blah blah blah blah blah...
```



What happens here?

```
t←'good day to you'
```

```
('g.+d' ⍋R 'X') t
```

Xay to you

```
('g.+d' ⍋R 'X' ⍋ 'Greedy' 0) t
```

X day to you



What happens here?

```
t←'good day to you'
('o' ⍳R 'X') t
gXXd day tX yXu
('o' ⍳R 'X' ⍳ 'ML' 2) t
gXXd day to you
('o' ⍳R 'X' ⍳ 'ML' -3) t
good day tX you
```



What happens here?

```
t←'good day to you'
('o' ⍳R 'X' ⍳ 'ML' 2) t t
```

gXXd day to you	gXXd day to you
-----------------	-----------------

```
('o' ⍳R 'X' ⍳ ('ML' 2)('Mode' 'D')) t t
```

gXXd day to you	good day to you
-----------------	-----------------



Search

```
t ← 'good day to you'
('o' ⍳ 'X') t
```

X	X	X	X
---	---	---	---

```
(' \b\w*d\w*\b' ⍳ '\0') t
```

good	day
------	-----



Search

t ← 'good day to you'

('DAY' '\w+' ⎕S 0 1 2 3 ⎕ 1) t t

0	4	0	1	5	3	0	0	9	2	0	1	12	3	0	1	0	4	1	1	5	3	1	0	9	2	1	1	12	3	1	1
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---

Start

Length

Block

Pattern



Search

t ← 'good day to you'

('DAY' '\w+' ⎕S 0 1 2 3 ⎕ 1) t t

0	4	0	1	5	3	0	0	9	2	0	1	12	3	0	1	0	4	1	1	5	3	1	0	9	2	1	1	12	3	1	1
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---

Start

Length

Block

Pattern



Search

t ← 'good day to you'

('DAY' ' \w+' ⍳ 0 1 2 3 ⍴ 1) t t

0	4	0	1	5	3	0	0	9	2	0	1	12	3	0	1	0	4	1	1	5	3	1	0	9	2	1	1	12	3	1	1
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---

Start

Length

Block

Pattern ...



What happens here?

```
t←'good day to you'  
( '\w+' ⌈S {w.Match}) t
```

good	day	to	you
------	-----	----	-----



What happens here?

```
t←'good day to you'
( '\w+' ⌈R {ω.(Match,':',⌈ρMatch)} ) t
good:4 day:3 to:2 you:3
```



What happens here?

```
t←'good day to you'  
⊖( '\w+' ⍲ {⊖ω.Match}) t  
you to day good
```



Tip

```
( '.+' ⍋R ⍋JSON) 'good'
```

```
{ "Block": "good", "BlockNum": 0, "Lengths": [4], "Match": "good", "Names": [""], "Offsets": [0], "Pattern": ".+", "PatternNum": 0, "ReplaceMode": 1, "TextOnly": 1 }
```



What happens here?

```
t←'the room size is 7 by 3'
```

```
m←3.28084
```

```
( '\d+' ⌈R {⌘m×⌘ω.Match}) t
```

```
the room size is 22.96588 by 9.84252
```



What happens here?

```
t←'1. the room size is 7.5m by 3.2m'
```

```
m←3.28084
```

```
('\'b([\d\.]*)m\' ⍲R {'ft',~⌈m×⌊ω.Lengths[2]↑ω.Offsets[2]↓ω.Block}) t
```

```
1. the room size is 24.6063ft by 10.498688ft
```



Tips

```
t←1000ρ'Good day to you'
```

```
f1←'oo' ⍲R 'X'
```

```
f2←'o{2}' ⍲R 'X'
```

```
cmpx 'f1 t' 'f2 t'
```

```
f1 t → 5.9E-6 | 0% ⍲
```

```
f2 t → 4.7E-5 | +693% ⍲⍲⍲⍲⍲⍲⍲
```



Tips

```
('Any text.' ⍲R 'X') 'Any text?'
```

X

18.0

```
('Any text.' ⍲R 'X' ⍲ 'Regex' 0) 'Any text?'
```

Any text?

```
('⍲Any text.⍲E' ⍲R 'X') 'Any text?'
```

Any text?



Tips

```
('vec' ⍲ 'X') 'vec1' 'vec2'
```

X1	X2
----	----

```
('.'+' ⍲ 'X' ⍲('Mode' 'D'))('DotAll' 1)) 'vec1' 'vec2'
```

X

```
('.'+' ⍲ 'X\rX') 'vec1' 'vec2'
```

X	X	X	X
---	---	---	---

```
('.'+' ⍲ 'X\rX' ⍲('Mode' 'D'))⍲'vec2' 'vec2'
```

X	X
X	X

Character
vectors in a
vector are *not*
independent



Tips

```
('X' ⍳R '\n') 'Hello There' 'GoodXThere'
```

Hello There	Good	There
-------------	------	-------

```
('X' ⍳R '\n') 'GoodXThere'
```

Good
There

```
('X' ⍳R '\n' ⍳ 'ResultText' 'Nested') 'GoodXThere'
```

Good	There
------	-------





Bonus exercise: count the words

t ← 'Rex forex r#e#x r#e##x R###E###X'

Rex:

- the letters 'r', 'e', 'x'
- in any case combination
- optionally with a sequence of '#'s between the 'r' and 'e' and the same between the 'e' and 'x'

Other: all other words



Bonus exercise: count the words

```
t ← 'Rex forex r#e#x r#e##x R###E###X'
```

```
⊖ ← b ← (... ⊖S ...) t
```

```
0 1 0 1 0
```

```
... ⊖b
```

rex	3
other	2



Bonus exercise: count the words

```
t←'Rex forex r#e#x r#e##x R###E###X'
```

```
⊖←b←('r(#*)e\1x' '[\w#]+' ⊖S 3 ⍴ 1) t
```

```
0 1 0 1 0
```

```
... ⊖b
```

rex	3
other	2



Bonus exercise: count the words

```
t←'Rex forex r#e#x r#e##x R###E###X'
```

```
⊖←b←('r(#*)e\1x' '[\w#]+' ⊖S 3 ⍤ 1) t
```

```
0 1 0 1 0
```

```
'rex' 'other',⌵{≠ω}⍤b
```

rex	3
other	2

