A Decade of APL Extensions
Grouping and Processing Text

Nicolas Delcros
Partitioning: \(<\) and \(\subseteq\)

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

'ABCDEF' \(\leftrightarrow\) 'AB' 'CDEF'
\( \alpha \subset \omega \)  
Partitioned Enclose
- \( \alpha \) is boolean  
  \( \omega \) is cut at 1’s  
- May only discard leading items of \( \omega \) if \( \alpha \) has leading 0’s  
- High-rank: Return vector of arrays

\( \alpha \subseteq \omega \)  
Partition
- \( \alpha \) is non-negative integer  
  \( \omega \) is cut at increases of \( \alpha > 0 \)  
- Can discard arbitrary items of \( \omega \) with \( \alpha = 0 \)  
- High-rank: Return array of vectors
Left argument

• \( \alpha \leq \omega \): boolean \( \alpha \), cuts at 1’s
  \[
  \text{disp} \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ \subseteq \ 'ABCDEF'
  \]

\[
\begin{array}{c}
\text{AB} \\
\text{CDEF}
\end{array}
\]

• \( \alpha \leq \omega \): integer \( \alpha \), cut at increases of positive \( \alpha \)
  \[
  \text{disp} \ 1 \ 1 \ 3 \ 3 \ 3 \ 1 \ \subseteq \ 'ABCDEF'
  \]

\[
\begin{array}{c}
\text{AB} \\
\text{CDEF}
\end{array}
\]
Left argument

- Exercise: Convert left argument of ⊂ into left argument of ⊆
- Behaviour:
  \[
  \{…\} \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
  1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2
  \]
- Solution:
  \[\alpha{\leq}\omega \iff (+/\alpha){\leq}\omega \quad \forall \text{ for any boolean } \alpha\]
Left argument: discarding items

- $\alpha \preceq \omega$: all items of $\omega$ end up in the result excepted 0’s before the first 1 in $\alpha$
  
  ```apl
  ]disp 0 1 0 0 1 0 \preceq 'ABCDEF'
  ```

- $\alpha \subseteq \omega$: any item of $\omega$ can be discarded with $\alpha = 0$
  
  ```apl
  ]disp 0 1 1 0 1 1 \subseteq 'ABCDEF'
  ```
Left argument: discarding items

- Exercise: prevent any item of $\omega$ to disappear from $\alpha \subset \omega$
- Behaviour:
  $$\{...\} \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0$$
  $$1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0$$
- Solutions:
  $$\begin{align*}
  (1, 1 \downarrow \alpha) & \subset \omega \quad \text{A two “heavy” operations} \\
  (1 \@ \text{IO} \neg \alpha) & \subset \omega \quad \text{A better (in-place)}
  \end{align*}$$
Left argument

• Exercise: convert left argument of $\leq$ into left argument of $\subset$

• Solution:
impossible because $\subset$ cannot discard arbitrary items of $\omega$
Left argument

• Exercise: convert positive left argument of \( \leq \) into left argument of \( < \)

• Solution:

\[
\alpha \leq \omega \iff (1, 2 < / \alpha) \subset \omega \quad \forall \text{ for any } \alpha > 0
\]

\[
\alpha \leq \omega \iff (2 < / 0, \alpha) \subset \omega \quad \forall \text{ allow leading zeros}
\]
Left argument

• Exercise: rewrite $\leq$ with $\subset$
• Solution:
  1. Discard items where $\alpha=0$
  2. Convert positive left argument of $\leq$ into left argument of $\subset$

$$\{m \leftarrow \alpha > 0 \land (m/2 < \omega, \alpha) \subset (m/\omega)\}$$
Converting between ∈ and ⊆

- \( \{ \alpha \in \omega \} \leftrightarrow \{(+ \backslash \alpha) \subseteq \omega\} \)
- \( \{ \alpha \subseteq \omega \} \leftrightarrow \{ m \leftarrow \alpha > 0 \diamond (m/2 < \omega, \alpha) \subseteq (m/\omega)\} \)
Exercise: cut string at separators

- Exercise: cut string $\omega$ at separators $\alpha$ with $\subset$ or $\subseteq$
- Behaviour:

\[
\begin{array}{c}
\text{\}disp ' ,,!?' \{…\} 'Hello world, how are you?'}
\end{array}
\]

\[
\begin{array}{cccc}
\text{Hello} & \text{world} & \text{how} & \text{are} & \text{you}
\end{array}
\]

\[
\begin{array}{cccc}
\text{\}disp ' ,,!?' \{…\} 'Hello world, how are you?'}
\end{array}
\]

\[
\begin{array}{cccc}
\text{Hello} & \text{world} & \text{how} & \text{are} & \text{you}
\end{array}
\]
Exercise: cut string at separators with \( \subseteq \)

- **Solution:**

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

  \[
  \text{world , how are you ?}
  \]

  \[
  \text{disp ' .,!?'}(-\omega \in \alpha)\subseteq \omega)'\text{Hello world, how are you}'
  \]

  \[
  \text{Hello world how are you}
  \]

- **Problem:** inability to cut empty strings (easily)
Exercise: cut string at separators with \( \subset \)

- **Solution:**
  ```apl```
  ```
  ]disp ' .,!?'{(⍵∊⍺)⊂⍵}'Hello world, how are you?'
  ```
  ```
  world , how are you ?
  ```
  ```
  ]disp ' .,!?'{l←(⊃⍺),⍵ ⋄ 1↓¨(l∊⍺)⊂l}'Hello world, how are you?'
  ```
  ```
  Hello world how are you
  ```
  ```
  • **Problem:** catenation and \( 1 \downarrow \cdots \)
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 \( \omega \)
- \( \alpha \subseteq \omega \): result has rank of \( \omega \), items are vectors
- \( \subset \) performs better
  (fewer nested items for same amount of data)
** Partitioned Enclose **

- $\alpha \subset \omega$
- $\alpha$ is boolean
- $\omega$ is cut at 1’s
- May only discard leading items of $\omega$ if $\alpha$ has leading 0’s
- High-rank: Return vector of arrays

** Partition **

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

(search_patterns □R replace_patterns □ name_value_pairs ) input_document

(search_patterns □S result_values □ name_value_pairs ) input_document
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]*$'
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*))' OR 'NIGHT' '\u1{l2}' ⌷ 1) t

Good NIGHT To You
What happens here?

t←'good day to you'

'(\w)(\w*)' 'DAY' ▶R '\u1\l2' 'NIGHT' ▶ 1) t

Good Day To You
Exercise: an English to Doglish translator

```apl
12
A Decade of APL Extensions – Grouping and Processing Text

Exercise: an English to Doglish translator

\[ t \leftarrow 'OK, Ginger! I''ve had it! You stay out of the garbage! Understand, Ginger? Stay out of the garbage, or else!' \]

\[
(\ ... \\ OR \ ...\ )\ t
\]

blah GINGER blah blah blah blah blah blah blah blah blah blah blah blah blah blah GINGER blah blah blah blah blah blah blah blah blah...

#dyalog19
A Decade of APL Extensions – Grouping and Processing Text
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+' '$' OR 'GINGER' 'blah' '...' ¶ 1) t

blah GINGER blah blah blah blah blah blah blah blah blah blah
GINGER blah blah blah blah blah blah blah blah blah'

#dyalog19
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?

```apl
A ← 'good day to you'
('o' R 'X' I: 'ML' 2) A A
```

<table>
<thead>
<tr>
<th>gXXd day to you</th>
<th>gXXd day to you</th>
</tr>
</thead>
</table>

```apl
A ← 'good day to you'
('o' R 'X' I: ('ML' 2)('Mode' 'D')) A A
```

<table>
<thead>
<tr>
<th>gXXd day to you</th>
<th>good day to you</th>
</tr>
</thead>
</table>
Search

\[
t \leftarrow ' \text{good day to you}'
\]

\[
('o' \in \text{S 'X'}) \ t
\]

\[
\begin{array}{cccc}
X & X & X & X
\end{array}
\]

\[
('b\w*d\w*b' \in \text{S '0'}) \ t
\]

\[
\begin{array}{c}
good \\
day
\end{array}
\]
Search

t←'good day to you'

('DAY' 'w+' S 0 1 2 3 & 1) t t
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 \leftarrow \text{"good day to you"}
\$

\$('DAY' \ owners \ 'w+' \ [0 1 2 3] \ $1) \ t \ t
\$

\begin{array}{cccccccccccccccc}
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
\end{array}

Start Length Block Pattern
What happens here?

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

good day to you
What happens here?

t←'good day to you'
('\w+' R {ω.(Match,':',พรรMatch)}) t

good:4 day:3 to:2 you:3
What happens here?

t←'good day to you'
φ(\w+ OR {φω.Match}) t

you to day good
Tip

`('.+' ▶R ▶JSON) 'good'`

```json
{
  "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?

\[ t \gets '1. \text{the room size is 7.5m by 3.2m}' \]
\[ m \gets 3.28084 \]
\[ ('\b([\d.]+)m\b' \( \times \) \{ 'ft', 'm' \} \( \times \) \text{Lengths}[2] \( ↓ \text{Offsets}[2] \) \( ↓ \text{Block} \}) \) 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

('Any text.' ⊢R 'X' ⊢ 'Regex' 0) 'Any text?'

Any text?

('\QAny text.\E' ⊢R 'X') 'Any text?'

Any text?
Tips

('vec' ∘R 'X') 'vec1' 'vec2'

X1  X2

('.' ∘R 'X' ⍪('Mode' 'D')('DotAll' 1)) 'vec1' 'vec2'

X

('.' ∘R 'X\rX') 'vec1' 'vec2'

X X X X

('.' ∘R 'X\rX'⍨'Mode' 'D')⌽'vec2' 'vec2'

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' R 'ResultText' 'Nested') 'GoodXThere'  
Good There
Bonus exercise: count the words

\[ t \leftarrow \text{'Rex forex r#e#x r#e##x R###E###X'} \]

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
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rex</td>
<td>3</td>
</tr>
<tr>
<td>other</td>
<td>2</td>
</tr>
</tbody>
</table>
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 | 3   |
| other | 2 |
+-----+-----+
```
Bonus exercise: count the words

t←'Rex forex r#e#x r#e##x R###E###X'
new←(r(#*)e\1x' '[\w#]+' ⊥S 3 ⊥ 1) t

0 1 0 1 0

rex 'other',¬{≠ω}⊥b

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rex</td>
<td>3</td>
</tr>
<tr>
<td>other</td>
<td>2</td>
</tr>
</tbody>
</table>