How I won the APL Problem Solving Competition

Who? Andrea Piseri (andrea.piseri@gmail.com)

From? Università Degli Studi di Milano

When? 18 Oct 2023

About Me

Education

- Maths student at Università Degli Studi di Milano
- Abstract algebra
- Mathematical logic
- Computability

CS Hobbies

- wrote ~50 lines of Python at age 12
- learned C in University
- programming language enthusiast (C/C++, Rust, Haskell, APL, BQN)

My Journey With APL (and array programming)

My Journey With APL (and array programming)

First encounter: MATLAB in university

- Yes, it's technically an array language
- Interpreted, dynamic
- Weird syntax, this is nothing like C!

Back to C (for now)

- Experiments with competitive programming
- Appreciation for arrays!

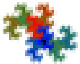
My Journey With APL (and array programming)

FP languages

- Haskell from Tsoding's community
- Composition patterns and combinators
- High information density
- Similar to math notation

Language comparisons

- Connor Hoekstra's Youtube channel
- Encountered APL
- *Higher* information density
- *Terser* math notation



My Journey With APL (and array programming)

Advent of Code

MATLAB's new chance!

December 2022

- learn APL by solving problems with it
- ∼1 new primitive per day
- published my solutions on **github** and **mastodon**
- finished days 1-20

The Competition Problems

The Competition Problems

Part 1 Bioinformatics

- Task 5: Reading frame translation
- “real world” programming
- two very different solutions

Part 2 Potpourri

- Task 3: Time for a Change
- induction!
- opportunity for improvement

Task 1.5: Reading frame translation

Task 1.5: Reading frame translation

Write a function that, given the name of a file in FASTA format, returns all the protein strings that can be translated from it, in all six reading frames.

FASTA format

```
>Rosalind_2748
ATCAGGCTACCGTGTTTGCGGACGGGGCTTAATCT
CTTGTTGGCACAGCGGTGGCAGGAGGTCCCCGCCGA
...
```

Output

```
'MVMATGVIVLNTRMRVTNDSNFGARYRGTCP' ...
 ... 'MGL' 'MDRL' 'MRLPWSCLHIA'
```

Final structure

```
orf ← {crf⍤aas⊃⌽⊃readFASTA⍵}
```

- `readFASTA` performs the IO
- `aas` converts from DNA to list of reading frames
- `crf` extracts the protein strings

Details

- `'$'` are stop codons.
- A protein string starts with `'M'`, ends before `'$'`.
- Any `'M'` not followed by `'$'` doesn't start a protein.

```
'XXXMAAM$YMBB'
```
becomes:
```
'MAAM' 'M'
```

```
'XXXMAAM$YMB$'
```
becomes:
```
'MAAM' 'M' 'MB'
```

First Implementation of `crf`

Main Idea

Solve for one sequence, map and flatten.

```
(a -> [b]) -> [a] -> [b]
```

```
⍝ utility: flat map modifier
      _f ← {⊃,/⍺⍺¨⍵}
      'abc'⍨_f ⍳5
'abcabcabcabcabc'
```

First Implementation of `crf`


```
[Amino] -> [[Amino]]
```

```
      ('$'∘≠⊆⊢) 'AA$BB$XX'
AA  BB  XX
      ((-'$'≠⊢/)↓'$'∘≠⊆⊢) 'AA$BB$XX'
AA  BB
```

- Split each sequence by `'$'`.
- Drop the last split of sequences that don't end in `'$'`.

First Implementation of `crf`

Suffixes starting with `'M'`

```
[Amino] -> [[Amino]]
```

```
   'M'∘(=⊂⊢) 'XXMAAAMBB'
 MAAA  MBBB
   'M'∘(,⍨\⍥⌽(=⊂⊢)) 'XXMAAAMBB'
 MBBB  MAAAMBBB
```

- `⊂` makes partitions starting with `'M'`.
- Drops anything before the first `'M'`.
- `,⍨\⍥⌽` concatenates the suffixes of the array.

Putting it all together

```
[[Amino]] -> [[Amino]]
```

```
crf ← {
    p ← ((-'$'≠⊢/)↓'$'∘≠⊆⊢)_fω
    ∪'M'∘(,⍨\⍤⌽(=⊂⊢))_f p
}
```

Second Attempt at `crf`

```
[[Amino]] -> ([Amino], [Int])
```

```
s d←(,/,(⊂(+\≠¨))),∘'$'¨'A$MB' 'C$MD$'
A$MB$M$MD$$    5 11
```

Second Attempt at `crf`

Finding Begins and Ends

```
s = 'A$MB$M$MD$$'
d = 5 11
```

```
  i←⊃{↑α(ω[1+ωɩ̲α])}/'M$'(ɩ̲=)¨⊂s
(3 6 8)  (2 5 7 10 11)
```

Second Attempt at `crf`

Finding Begins and Ends

```
s  =  'A$MB$M$MD$$'
d  =  5 11
```

```
    i←⊃{↑α(ω[1+ω⍸α])}/'M$'(⍸=)¨⊂s
    ⍺ α ≡ 3 6 8 ; ω ≡ 2 5 7 10 11
2  3  4
```

Second Attempt at `crf`

```
s  =  'A$MB$M$MD$$'
d  =  5 11
```

```
    i←⊃{↑α(ω[1+ωɩ̲α])}/'M$'(ɩ̲=)¨⊂s
3   6    8
5   7   10
```

```
    b e←¯1+↓(~i[2;]∊d)/i
(5 7) (6 9)
```

Second Attempt at `crf`

```
crf ← {
    s d←(,/,(⊂(+\≠¨))),∘'$'¨⍵
    i←⊃{↑⍺(⍵[1+⍵⍸⍺])}/'M$'(⍸=)¨⊂s
    b e←¯1+↓(~i[2;]∊d)/i
    ∪b↓¨e↑¨⊂s
}
```

Task 2.3: Time for a Change

Task 2.3: Time for a Change

Problem statement

Write a function that takes a list of denominations as a left argument and a total value as a right argument, and returns a matrix where each row represents a unique combination of the elements of the left argument that total the right argument.

In other words, find a non negative integer matrix `r` with unique rows, maximising `≢r` under the constraint `⍵∧.=r+.×⍺`

Task 2.3: Time for a Change

Pruning

$$\mathrm{GCD}(\alpha_1, \cdots, \alpha_n) | \omega \Leftrightarrow \exists \beta_1, \cdots, \beta_n \in \mathbb{Z},\ \omega = \sum_{i=1}^{n} \alpha_i \beta_i$$

Inductive base

```
0≠ω|⍨∨/α: (0,≢α)⍴0
1=≢α:       ⍪ω÷⊃α
```

- Avoid recursive calls
- Base case is trivial

Task 2.3: Time for a Change

Inductive step

```
α≡1 2 3 ⋄ ω≡7
```

```
      i←0,⍳⌊ω÷a←⊃⌽α
0 1 2
```

Task 2.3: Time for a Change

Solving subproblems

```
α≡1 2 3 ⋄ ω≡7 ⋄ i≡0 1 2 ⋄ (ω-a×i)≡7 4 1
```

```
      s←(⊂¯1↓α)∇¨ω-a×i
 7 0    4 0    1 0
 5 1    2 1
 3 2    0 2
 1 3
```

Task 2.3: Time for a Change

Merging step

```
     s,¨i
7 0 0   4 0 1   1 0 2
5 1 0   2 1 1
3 2 0   0 2 1
1 3 0
```

Task 2.3: Time for a Change

Final solution

```
makeChange ← {
     0≠⍵|⍨∨/⍺: (0,≢⍺)⍴0
     1=≢⍺:     ⍪⍵÷⊃⍺
     i←0,⍳⌊⍵÷a←⊃⌽⍺
     s←(⊂¯1↓⍺)∇¨⍵-a×i
     ⊃⍪/s,¨i
}
```

- Algorithmically better solution discussed on **my blog**

What I leaned

What I leaned

APL is not as opinionated as I thought.

- direct code translation can work
- functional patterns apply

Array Programming has been there all along

- APL techniques apply to MATLAB, C++, etc.
- It changes the way you think