Reduction Of Singletons

Bob Smith
Sudley Place Software
September 2010
Updated 5 Apr 2017

Despite the elegance of APL notation and its implementation, there are still a few dark corners. One that has bothered me for quite some time is illustrated by

```
+/'A' ←→ 'A'                                                    (1-1)
```

I find that result anomalous, out of range, and just plain ugly. Fortunately, there is a simple, short, and elegant solution.

The current definition of reduction on vectors when extended to handle nested arrays and arbitrary functions divides into three distinct cases[1]:

```
f/V ←→ ⊂(↑V) f ⊃f/1↓V        (for 1<⍴V)                        (1-2a)
    ←→ ⊂↑V                    (for 1=⍴V)                        (1-2b)
    ←→ ⊂IdentityElement       (for 0=⍴V)                        (1-2c)
```

The definition in (1-2b) is what produces the anomalous result (1-1). It defines the result of reduction on a singleton in a manner which allows the recursion in (1-2a) to terminate without regard as to the validity of the result. It's possible to achieve those two goals so as to allow a more natural definition of the result of reduction on a singleton as follows:

```
f/V ←→ ⊂(↑V)   f ⊃f/1↓V      (for 2<⍴V)                        (1-3a)
    ←→ ⊂(0⊃V) f 1⊃V          (for 2=⍴V)                        (1-3b)
    ←→ ⊂(↑V)   f ⊃f/1↓V      (for 1=⍴V)                        (1-3c)
    ←→ ⊂IdentityElement       (for 0=⍴V)                        (1-3d)
```

In other words, for the singleton case, combining (1-3c) and (1-3d) yields a definition for Reduction of Singletons:

```
f/V ←→ ⊂(↑V) f IdentityElement     (for 1=⍴V)                  (1-4a)
```

Note that for functions (such as <) whose identity element is left and not right, this equation is written as

```
f/V ←→ ⊂ IdentityElement f ↑V      (for 1=⍴V)                  (1-4b)
```

For example,

```
+/'A'  ↔  'A'  + 0 ↔         DOMAIN ERROR, previously 'A'            (1-5)
×/'A'  ↔  'A'  × 1 ↔         DOMAIN ERROR, previously 'A'            (1-6)
=/'A'  ↔  'A'  = 1 ↔   0     Previously 'A'                          (1-7)
≠/'A'  ↔  'A'  ≠ 0 ↔   1     Previously 'A'                          (1-8)
⊛/'A'  ↔                     DOMAIN ERROR -- no identity element for ⊛ (1-9)
+/1.1  ↔  1.1  + 0 ↔ 1.1                                             (1-10)
×/1.1  ↔  1.1  × 1 ↔ 1.1                                             (1-11)
=/1.1  ↔  1.1  = 1 ↔   0     Previously 1.1                          (1-12)
=/1    ↔   1   = 1 ↔   1                                             (1-13)
=/0    ↔   0   = 1 ↔   0                                             (1-14)
≠/1.1  ↔  1.1  ≠ 0 ↔   1     Previously 1.1                          (1-15)
≠/1    ↔   1   ≠ 0 ↔   1                                             (1-16)
≠/0    ↔   0   ≠ 0 ↔   0                                             (1-17)
```

Not all of these results might be obvious at first, but at least they are logical and in range.

One other consequence of this definition is that both `,/1` and `,/,1` signal a `DOMAIN ERROR` because `,/θ` does as per (1-3c). That is, although catenate has an associated identity function, it is not defined in this case. Starting with (1-3c), here's a proof:

```
f/V ↔ ⊂(↑V) f ⊃f/1↓V          (for 1=ρV)                             (1-3c)
```

and substituting these parameters yields

```
,/1   ↔  ,/,1                                                        (1-18)
,/,1  ↔  ⊂(↑,1) , ⊃,/1↓,1                                            (1-19)
      ↔  ⊂1 , ⊃,/1↓,1                                                (1-20)
      ↔  ⊂1 , ⊃,/θ                                                   (1-21)
```

Now if `,/θ` is an identity element, then

```
⊂1    ↔  ⊂1 , ⊃,/θ                                                   (1-22)
 1    ↔   1 , ⊃,/θ                                                   (1-23)
```

where we see that there is no value for `⊃,/θ` that can satisfy (1-23) because the scalar on the left side can never be the result of a catenation on the right side. That is, **Scalars are not in the range of catenate**. In general, `,/V` for `0=ρV` is not defined if the prototype of `V` is a scalar.

In order to get a result from `,/V` where `0=ρV`, we need to ensure that `V` has some depth such as in `⊂θ ↔ ,/0ρ⊂ι3` because `θ` is the identity element for catenation of vectors, such as `ι3`.

Other Instances of Reduction

The idea of reduction appears in other derived functions such as scan, N-wise reduction, and inner product and those definitions must be reviewed in the light of this change.

Scan is repeated reduction which at the left edge reduces a singleton, and is computed according to the rules above. In particular,

```
0⊃ f\ V ↔ ⊃f/1↑V                    (for 0<⍴V)                 (2-1)
        ↔ (↑V) f IdentityElement    (from (4))                 (2-2)
```

A consequence of this definition is that if the left operand does not have an identity element (or identity function, or the identity function is not defined on the singleton), scan signals a `DOMAIN ERROR`. For example,

```
,\ 1 2 3 ↔ DOMAIN ERROR                                        (2-3a)
```

because as shown above starting with (1-18) catenate reduction is not defined on a one-element vector whose item is a scalar; previously, this expression yielded

```
1 (1 2) (1 2 3)                                                (2-3b)
```

On the other hand,

```
,\,¨ 1 2 3 ↔ (,1) (1 2) (1 2 3)                                (2-3c)
```

which has an advantage over the previous result as it is of uniform depth.

N-wise Reduction (sometimes called dyadic reduction) with a left argument of `±1` defines the window over which reduction occurs as a collection of singletons, and is computed according to the rules above. In particular,

```
1 f/ V ↔ ⊃¨f/¨⊂¨V                                              (2-4a)
       ↔ ⊃∘(f/)∘⊂¨V                                            (2-4b)
       ↔ f/⍢⊂¨V                 where ⍢ is the Dual Operator   (2-4c)
```

For example,

```
1 </ 1 2 3 4 ↔ </⍢⊂¨   1 2 3 4                                 (2-5a)
             ↔ ⊃¨</¨⊂¨ 1 2 3 4                                 (2-5b)
             ↔ ⊃¨</¨   1 2 3 4                                 (2-5c)
             ↔ ⊃¨      1 1 1 1                                 (2-5d)
             ↔         1 1 1 1                                 (2-5e)
```

Note that the result in line (2-5d) follows from (1-4b) (not (1-4a)) as per the following example:

```
</2  ↔ ⊂(</⍬)<↑2                                                    (2-6a)
     ↔ ⊂(</⍬)< 2                                                    (2-6b)
     ↔ ⊂  0  < 2                                                    (2-6c)
     ↔ ⊂1                                                           (2-6d)
     ↔ 1                                                            (2-6e)
```

Previously, this expression yielded `1 2 3 4`.

Inner Product involves a comparison function followed by a reduction function. When both inner dimensions are of length one, the reduction is of a singleton, and is computed according to the rules above. In particular, where both inner dimensions are of length one, and

```
L1←(¯1↓⍴L)⍴L ⋄ R1←(1↓⍴R)⍴R                                          (2-7)
```

```
L f.g R ↔ ⊃¨f/¨⊂¨L1∘.g R1                                           (2-8)
        ↔ ⊃∘(f/)∘⊂¨L1∘.g R1                                         (2-9)
        ↔ f/⍢⊂¨L1∘.g R1        where ⍢ is the Dual Operator         (2-10)
```

System Labels

If you implement System Labels[2], then using the above definitions for a user-defined function `foo` and singleton `S`, `foo/S` is defined as

```
⊂(↑S) foo ⊃foo/0⍴S                                                  (3-1)
```

which has the effect of invoking `foo` twice: once at the label `⎕ID` to produce the function's identity element and then again at line 1 to produce the result.

For example,

```
    ∇ Z←L plus R
[1]   Z←L+R
[2]   L '+' R '→' Z ⋄ →0
[3]   ⎕ID:'⎕ID: ',Z←+/0⍴L
    ∇
      plus/8
⎕ID:  0
8 + 0 → 8
8
```

Extended Monadic Iota

In the NARS2000 APL interpreter and elsewhere, this function has been extended to multielement vectors via the expression `□∘.,/⍳¨V`, where we presume that monadic iota is initially defined on integer scalars only.

```
      ⍳2 3
 1 1   1 2   1 3
 2 1   2 2   2 3
```

What happens with this expression when `V` is a one-element vector (say, `V←,3`)? That is, in the limiting case of a one-element vector, does this expression give us the expected result of `⍳,3 ←→ ⍳3`?

From equation (1-3c) above with `f←∘.,`, we have

```
∘.,/V ←→ ⊂(↑V) ∘., ⊃∘.,/1↓V   (for 1=⍴V)        (1-3c)
ID ←→ ∘.,/0⍴V                                   (4-1)
↑V ←→ (↑V) ∘., ⊃ID                              (4-2)
```

This means that `⊃ID` must be a scalar, otherwise the outer product will produce the wrong shape.

Choosing a typical element from (4-2)

```
I⊃↑V ←→ I⊃(↑V) ∘.,  ⊃ID                         (4-3)
     ←→ (I⊃↑V)   , ⊃⊃ID                         (4-4)
```

Here, too, we see that if any item of `↑V` is a scalar, there is no solution because **Scalars are not in the range of catenate**.

Now back to the extension of monadic iota to integer vectors,

```
⍳V ←→ ⊃∘.,/⍳¨V
```

For the limiting case `⍳¨,3` where we wondered if it produces `⍳3` when using the new interpretation of reduction of singletons

```
⊃∘.,/⍳¨,3 ←→ ⊃⊂(↑⍳¨,3) ∘., ⊃∘.,/1↓⍳¨,3
```

The identity element part of the above equation is

```
⊃∘.,/1↓⍳¨,3 ←→ ⊃∘.,/1↓1⍴⊂⍳3
            ←→ ⊃∘.,/1↓1⍴⊂1 2 3
            ←→ ⊃∘.,/0⍴⊂1 2 3
```

which fits equation (4-1) above for `V←1⍴⊂1 2 3` which reduces to equation (4-4) which we now see has no solution because `↑V ←→ 1 2 3` has (all) scalar items. Thus,

```
⊃∘.,/⍳¨,3 ←→ DOMAIN ERROR
```

This means that `⊃∘.,/⍳¨V` works for multiple element integer vectors only; otherwise it signals a `DOMAIN ERROR`. So we are free to substitute a result for the `DOMAIN ERROR` when `V` is a one-element integer vector which we do with `⍳θρV`.

Finally, there is the empty case

```
⊃∘.,/⍳¨θ ←→ ⊃∘.,/0ρ⊂⍳0
```

which also fits equation (4-1) for `V←1ρ⊂θ` which reduces to equation (4-4).

```
I⊃↑V ←→ (I⊃↑V) , ⊃⊃ID
```

which signals a `DOMAIN ERROR` because `↑V ←→ θ` and an item of `θ` (that is `0 ←→ ↑θ`) is a scalar, a result I'm happy to leave as an error.

References

1. J. A. Brown , M. A. Jenkins, “The APL Identity Crisis”, APL81, p.62-66.
2. David A. Rabenhorst, “APL Function Variants and System Labels”, APL83, p.281-284.