ALL Notation and its Properties

Andre Bhattacharyya

2-4-25

1 Abstract

Describing or showing many repeated operations on lists of values can be tricky. These nests of operations and operands can be too long to describe perfectly, but compressing their expressions could lead to confusion. However, with a new notation, we can describe these complex folds/reductions in a short notation that leaves no room for misconceptions by having a perfect algorithm to execute it. In this document, we shall refer to this notation as ALL notation. With this ALL notation, we can describe summation, products, continued fractions, and infinitely many more folds/reductions with ease and accuracy. To respect its name and what it can represent, we will write the notation with a large ALL, bounds like summation or products, and an array containing the operations and operands.

2 Introduction to ALL Notation

In this document, we will explore the usage and generalization of operations on (infinite) lists of values. Let the notation:

$$\overset{x}{\underset{n=1}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix}$$

Equate to:

$$a_{1,1}\star_1(a_{1,2}\star_2(\cdots a_{1,y-1}\star_{y-1}(a_{1,y}\star_y(a_{2,1}\star_1(a_{2,2}\star_2(\cdots a_{2,y-1}\star_{y-1}(a_{2,y}\star_y(\cdots a_{x,1}\star_1(a_{x,2}\star_2(\cdots a_{x,y-1}\star_{y-1}a_{x,y}))))))))))$$

Which can be compacted to:

$$\overset{x}{\underset{n=r}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix} = a_{r,1}\star_1(a_{r,2}\star_2(\cdots a_{r,y-1}\star_{y-1}(a_{r,y}\star_y(\overset{x}{\underset{n=r+1}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix}))))$$

Where:

$$\overset{x}{\underset{n=x}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix} = a_{x,1} \star_1 (a_{x,2} \star_2 (\cdots a_{x,y-1} \star_{y-1} (a_{x,y})))$$

Where $\star_n$ denotes a binary operation and $a_{b,c}$ is the bth cth value of a. The nonstandard matrix of our ALL notation is an argument of the function with a datatype of an array. The index and upper bound of our notation are obviously integers like in other notations such as summation. This is also known as a fold or reduction in functional programming. The notation above specifically folds right with multiple operators and operands. Our notation describes what we will call in this paper a(n) (multi-)operational series. The term does not refer to simply multiple operations in summation notation in this paper. This notation implies:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ + \end{bmatrix} = \sum_{n=1}^{\infty} a_n$$

And

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ * \end{bmatrix} = \prod_{n=1}^{\infty} a_n = \underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ \times \end{bmatrix}$$

And

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n & b_n \\ / & + \end{bmatrix} = \underset{n=1}{\overset{\infty}{\mathrm{K}}} \frac{a_n}{b_n} = \underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n & b_n \\ \div & + \end{bmatrix}$$

We can use more than what we consider standard binary operations. For instance:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ = \end{bmatrix} = (a_1 = a_2 = a_3 = \cdots)$$

Or

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} A_n \\ \bigcup \end{bmatrix} = (A_1 \bigcup A_2 \bigcup A_3 \bigcup \cdots)$$

Or

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} A_n \\ \vee \end{bmatrix} = (A_1 \vee A_2 \vee A_3 \vee \cdots)$$

This is because binary relations are also acceptable as inputs for binary operators. However, because there are parentheses around the next values of our ALL notation, we must specify in specific scenarios that:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ = \end{bmatrix} = (a_1 = (a_2 = (a_3 = (\cdots))))$$

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} A_n \\ \vee \end{bmatrix} = (A_1 \vee (A_2 \vee (A_3 \vee (\cdots))))$$

It is important to notice that when given:

$$\overset{x}{\underset{n=r}{ALL}} \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix}$$

The expression above should have $(x-r+1)y$ operands and $(x-r+1)y-1$ binary operations when fully expanded. y in this scenario will represent the number of columns of the array argument of the notation. This notation usually takes two types of data types as inputs in the array (variables (values that can be any datatype (usually numbers)) on the top row of the nonstandard matrix (the array) and binary operations on the bottom (functions or relations between/of two values)). The number of binary operation arguments in the bottom row of the array must equate to the number of variable arguments in the top row of the array. Based on the inputs, we can get various data types as outputs for these ALL notations. Because binary operations are simply functions between two operands, we can describe binary operations as a function $f(x, y)$ as shown below.

Let:

$$a \star_f b = f(a, b)$$

Therefore, given:

$$a + b = a \star_+ b$$

We get:

$$a + b = a \star_+ b = +(a, b)$$

Therefore:

$$\overset{3}{\underset{n=1}{ALL}} \begin{bmatrix} a_n \\ + \end{bmatrix} = a_1 + (a_2 + (a_3)) = a_1 + (+(a_2, a_3)) = +(a_1, +(a_2, a_3))$$

Or in general:

$$\overset{3}{\underset{n=1}{ALL}} \begin{bmatrix} a_n \\ \star_f \end{bmatrix} = a_1 \star_f (a_2 \star_f (a_3)) = a_1 \star_f (f(a_2, a_3)) = f(a_1, f(a_2, a_3))$$

We can even generalize this further to two or more functions:

$$\overset{3}{\underset{n=1}{ALL}} \begin{bmatrix} a_n & b_n \\ \star_f & \star_g \end{bmatrix} = a_1 \star_f (b_1 \star_g (a_2 \star_f (b_2 \star_g (a_3 \star_f b_3))))$$

Where the equality above equates to:

$$f(a_1, g(b_1, f(a_2, g(b_2, f(a_3, b_3)))))$$

Similar to summation notation and other notations, we can set the index of our ALL notation to a number other than one or a set. For instance:

$$\underset{n\in\{1,3,5\}}{ALL}\begin{bmatrix}a_n\\+\end{bmatrix} = a_1 + a_3 + a_5$$

Interestingly:

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n & b_n\\+ & +\end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n + b_n\\+\end{bmatrix}$$

And

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n & b_n\\* & *\end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n * b_n\\*\end{bmatrix}$$

Because of the properties:

$$a + b + c + d = (a + b) + (c + d)$$

$$a * b * c * d = (a * b) * (c * d)$$

Usually:

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n & b_n\\= & =\end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}a_n = b_n\\=\end{bmatrix}$$

And:

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}A_n & B_n\\\bigcup & \bigcup\end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}A_n \bigcup B_n\\\bigcup\end{bmatrix}$$

The notation with equalities above is usually true because an equality returns a boolean and if $a = b = c = d$ is true, $a = b$ is true and $c = d$ is true, implying $(a = b) = (c = d)$ because true = true. However, if $a = b = c = d$ is false, $(a = b) = (c = d)$ may return true. However, both statements usually are false for randomized inputs. As you can see, subtraction, division, and many other operations would differ:

$$a - b - c - d \neq (a - b) - (c - d)$$

$$a/b/c/d \neq (a/b)/(c/d)$$

Where a, b, c, d are different arbitrary values. On another tangent, we can attempt to define our ALL notation with some basic programming. Let:

$$All(A, B, C, x, y) = \overset{y}{\underset{n=x}{ALL}}\begin{bmatrix}C[1](A[1], n) & C[2](A[2], n) & \cdots & C[z](A[z], n)\\ \star_{B[1]} & \star_{B[2]} & \cdots & \star_{B[z]}\end{bmatrix}$$

Where A is a list of values (operands), B and C are lists of two-input functions, x and y are integers, and $z = \#A = \#B = \#C$. Function All is could be defined as:

```
local function All(A, B, C, x, y)
local Output = 0
if #A==#B and #B==#C then
for i = y, x, -1 do
for ii = #B, 1, -1 do
Output = B[ii](C[ii](A[ii],i), Output)
end
end
return Output
end
end
```

However, the program above would be perfect if the result was not initially set to 0. The result should be initially set to the last value, however that would complicate the for-loops. Executing this script with binary operations such as a product would reveal errors like it resulting in zero when the actual ALL notation is not zero. Perhaps we can solve this problem by improving the code above:

```
local function All(A, B, C, x, y)
local Output = 0
if #A==#B and #B==#C then
for i = y, x, -1 do
for ii = #B, 1, -1 do
if ii == #B and i == y then
Output = C[#B](A[#B],y)
else
Output = B[ii](C[ii](A[ii],i), Output)
end
end
end
return Output
end
end
```

On yet another tangent, we can also input functions in the top row of the nonstandard matrix of our ALL notation. For instance:

$$(\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} f_n \\ \circ \end{bmatrix})(y) = (f_1 \circ f_2 \circ f_3 \circ \cdots \circ f_x)(y)$$

Or in general:

$$(\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} f_n \\ \star_n \end{bmatrix})(x) = (f_1 \star_1 (f_2 \star_2 (f_3 \star_3 (\cdots f_{x-1} \star_{x-1} f_x \cdots))))(y)$$

Where f_a is a function and $\circ$ denotes the function composition operation. We can also define or use new binary operations that may reveal interesting properties of our ALL notation. Let:

$$a \star_{\text{return first}} b = a$$

$$a \star_{\text{return second}} b = b$$

$$a\%b = \text{mod}(a, b)$$

$$a \star_{\text{BesselI}} b = \text{BesselI}(a, b) = I_a(b)$$

Acknowledge:

$$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a \\ \star_{\text{return first}} \end{bmatrix} = a = \overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a \\ \star_{\text{return second}} \end{bmatrix}$$

$$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a & b \\ \star_{\text{return first}} & \star \end{bmatrix} = a$$

$$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a & b \\ \star_{\text{return second}} & \star \end{bmatrix} = b \star (\overset{x}{\underset{n=2}{ALL}} \begin{bmatrix} a & b \\ \star_{\text{return second}} & \star \end{bmatrix}) = \overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} b \\ \star \end{bmatrix}$$

Having repeated modulus may also reveal some interesting properties:

$$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a \\ \% \end{bmatrix} = \begin{matrix} a & x = 1 \\ 0 & x = 2 \\ \text{undefined} & x \neq 1, 2 \end{matrix}$$

$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a & b \\ \% & \% \end{bmatrix}$ is undefined for many rational inputs when $x > 1$

If a and/or b are different irrational values, the notation will likely not be undefined for most if not all x. As x approaches ∞, the ALL notation will likely

either be undefined or converge to 0. The reason why these may be undefined is because $a\%0$ is undefined and $a\%a = 0$. Moving on to the BesselI operation, let:

$$g(x) = \overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a_n \\ \star_{\text{BesselI}} \end{bmatrix} = I_{a_1}(I_{a_2}(\cdots I_{a_{x-1}}(a_x)\cdots))$$

Acknowledging:

$$I_x(y) = \sum_{k=0}^{\infty} \frac{2^{-2k-x}y^{2k+x}}{k!\Gamma(1+k+x)}$$

We get:

$$g(x) = \sum_{k=0}^{\infty} \frac{2^{-2k-a_1}(\overset{x}{\underset{n=2}{ALL}} \begin{bmatrix} a_n \\ \star_{\text{BesselI}} \end{bmatrix})^{2k+a_1}}{k!\Gamma(1+k+a_1)}$$

Or in general:

$$\overset{x}{\underset{n=q}{ALL}} \begin{bmatrix} a_n \\ \star_{\text{BesselI}} \end{bmatrix} = \sum_{k=0}^{\infty} \frac{2^{-2k-a_q}(\overset{x}{\underset{n=q+1}{ALL}} \begin{bmatrix} a_n \\ \star_{\text{BesselI}} \end{bmatrix})^{2k+a_q}}{k!\Gamma(1+k+a_q)}$$

This concludes the (arbitrarily) chosen list of binary operations to analyze. Although there are infinitely many binary operations possible, finding new and valid binary operations that cannot be defined algebraically by other binary operations or two-input functions can be difficult. Perhaps an interesting one is concatenation ($||$), where if a natural number a is concatenated with a natural number b, we get:

$$a||b = a * 10^{\text{floor}(\log(10b))} + b, a, b \in \mathbb{N}$$

However, decimals may also be acceptable in concatenation, extremely complicating finding an equality to it for both rational and natural inputs of a and b. One of the reasons why this is difficult is because the concatenation must take into account that decimals in some number bases may have an infinite string of numbers ($1/3 = 0.3333\cdots$) whereas in other bases, the string is finite ($1_3/10_3 = 0.1_3 = 0.3333\cdots_{10}$). This along with numerous other factors makes the concatenation operation perhaps impossible to describe with our standard binary operations in an algebraic two-input function. Moving on, perhaps we can define some other interesting notations using the ALL notation. Acknowledge the general form for branched continued fractions (BCFs) is:

$$d_{r_{x-1}}(z) + \overset{y}{\underset{k=x}{\text{D}}} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)} = d_{r_{x-1}}(z) + \sum_{i_x=r_x}^{N_x} \frac{c_{i_x}(z)}{d_{i_x}(z) + \overset{y}{\underset{k=x+1}{\text{D}}} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)}}$$

Ignoring the function outside the CF, we get:

$$\mathop{\mathrm{D}}_{k=x}^{y} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)} = \sum_{i_x=r_x}^{N_x} \frac{c_{i_x}(z)}{d_{i_x}(z) + \mathop{\mathrm{D}}_{k=x+1}^{y} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)}}$$

Unfortunately, it appears that using our standard form of our ALL notation cannot replicate the notation above. On a side-note, can redefine n-ary operations in the following way:

$$\left[\star_f \left\{ \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{array} \right. \right] = f(a_1, a_2, a_3, \cdots, a_n)$$

For example:

$$\left[+ \left\{ \begin{array}{c} a_1 \\ a_2 \end{array} \right. \right] = +(a_1, a_2) = a_1 + a_2$$

We can also nest these functions in each other:

$$\left[+ \left\{ \begin{array}{cc} & a_1 \\ \star \left\{ \begin{array}{c} a_2 \\ a_3 \end{array} \right. & \end{array} \right. \right] = +(a_1, \star(a_2, a_3)) = a_1 + (a_2 \star a_3)$$

This notation is great because it clearly shows how many operands each n-ary operation uses. Using the notation above with our ALL notation, we could get:

$$\mathop{ALL}_{n=1}^{x} \begin{bmatrix} a_n \\ \star \end{bmatrix} = \left[\star \left\{ \begin{array}{cc} & a_1 \\ \star \left\{ \begin{array}{cc} & a_2 \\ \cdots \star \left\{ \begin{array}{c} a_{x-1} \\ a_x \end{array} \right. & \end{array} \right. & \end{array} \right. \right]$$

This is just one of the many ways to describe the ALL notation in this document.

3 Various Properties of ALL Notation

In this section, we will explore the various properties of our ALL notation with specific inputs. Recall:

$$\mathop{ALL}_{n=1}^{\infty} \begin{bmatrix} a_n & b_n \\ + & + \end{bmatrix} = \mathop{ALL}_{n=1}^{\infty} \begin{bmatrix} a_n + b_n \\ + \end{bmatrix}$$

And

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n & b_n \\ * & * \end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n * b_n \\ * \end{bmatrix}$$

Because of the properties:

$$a + b + c + d = (a + b) + (c + d)$$

$$a * b * c * d = (a * b) * (c * d)$$

Therefore, given $\star$ is a binary operation where:

$$a \star (b \star (c \star (d))) = (a \star b) \star (c \star d)$$

Then:

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n & b_n \\ \star & \star \end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n \star b_n \\ \star \end{bmatrix}$$

We can also describe this property with our function notation of binary operations. Given $a \star_f b = f(a, b)$ and $f(f(a, b), f(c, d)) = f(a, f(b, f(c, d)))$ then:

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n & b_n \\ \star_f & \star_f \end{bmatrix} = \overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_n \star_f b_n \\ \star_f \end{bmatrix}$$

Functions such as $f(x, y) = x + y$ and $f(x, y) = x * y$ satisfy the formula above. $f(x, y) = x - y$, $f(x, y) = x/y$, $f(x, y) = x^y$, $f(x, y) = x\%y$, etc. do not satisfy the formula above. Some more functions that DO satisfy the formula above are:

$$f(x, y) = x$$
$$f(x, y) = z, z \in \mathbb{C}$$
$$f(x, y) = y$$
$$f(x, y) = |x|$$
$$f(x, y) = |y|$$
$$f(x, y) = \lfloor x \rfloor$$
$$f(x, y) = \lceil x \rceil$$
$$f(x, y) = \lfloor x \rceil$$
$$f(x, y) = \lfloor y \rfloor$$
$$f(x, y) = \lceil y \rceil$$
$$f(x, y) = \lfloor y \rceil$$

There are some more functions that satisfy the ALL notation equality (such as the sign functions of x and y), but many simplify to the functions above and

sometimes the addition and multiplication functions. Finding functions that use both x and y in their equality, satisfy the ALL notation equality, and do not simplify to any of the functions above can be extremely difficult beyond addition and multiplication. One that works for rational numbers with finite decimal expansions is the concatenation function:

$$f(x, y) = x||y$$

Of course, if x has an infinite decimal expansion, the concatenation does not work. Therefore, addition and multiplication are perhaps the two outliers of almost all binary operations in this way. Concatenation comes close to addition and multiplication, but only is valid for natural number inputs. Multiplication and addition on the other hand, accept all number value inputs. There are, of course, numerous datatypes in mathematics. Our ALL notation accepts almost all datatypes as value inputs in the top row of the array argument. The concatenation operation also works with regular strings. In fact, the concatenation operation satisfies the ALL notation equation above with string inputs. Now we shall see more properties of the ALL notation with specific inputs. It is true that:

$$2 \overset{\overset{\infty}{ALL}}{\underset{n=1}{}} \begin{bmatrix} 2 & 2 \\ / & \uparrow \end{bmatrix} = 2 = \overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} 2 & 2 \\ \uparrow & / \end{bmatrix}$$

Where $\uparrow$ denotes the exponent operation. In fact, it is true that:

$$a \overset{\overset{\infty}{ALL}}{\underset{n=1}{}} \begin{bmatrix} b & b \\ / & \uparrow \end{bmatrix} = a$$

This is because every iteration of the ALL notation simply makes the exponent 1. This implies:

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} b & b \\ / & \uparrow \end{bmatrix} = 1$$

Interestingly:

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a & a \\ \uparrow & / \end{bmatrix} = a \text{ if } a \in [1/e, 2]$$

The graph of the notation above looks quite interesting with linear and then exponential parts across specific bounds. Outside the bounds of $[1/e, 2]$ for a, the notation diverges. Therefore, the domain of the notation above is in fact $a \in [1/e, 2]$. Negative integer inputs for a also appear to converge to 0 (negative non-integers are complex and may diverge). This implies the domain is actually:

$$D_f : x \in \bigcup_{n=1}^{\infty} \{-n\} \bigcup [1/e, 2]$$

Where:

$$f(x) = \underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} x & x \\ \uparrow & / \end{bmatrix}$$

Perhaps something more interesting would be using both addition and multiplication as the binary operation arguments of ALL notations. Can we simplify the expression:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n & b_n \\ + & * \end{bmatrix}$$

In terms of summation and product notation? The notation above equates to:

$$a_1 + b_1 * (a_2 + b_2 * (\cdots))$$

Which can be rewritten as:

$$a_1 + a_2 b_1 + a_3 b_2 b_1 + \cdots$$

Which equates to:

$$a_1 + \sum_{n=1}^{\infty} (a_{n+1} \prod_{k=1}^{n} b_k)$$

However, each iteration of our ALL notation gives:

$$a_1 + b_1 * (a_2 + b_2 * (\cdots a_n + b_n))$$

Which actually equates to:

$$a_1 + a_2 b_1 + a_3 b_1 b_2 + \cdots + (\prod_{k=1}^{n-1} b_k)(a_n + b_n)$$

Which equals:

$$a_1 + (\prod_{k=1}^{n} b_k) + \sum_{k=2}^{n} (a_k \prod_{m=1}^{k-1} b_m)$$

Therefore:

$$\lim_{x \to \infty} \underset{n=1}{\overset{x}{ALL}} \begin{bmatrix} a_n & b_n \\ + & * \end{bmatrix} = \lim_{x \to \infty} a_1 + (\prod_{k=1}^{x} b_k) + \sum_{k=2}^{x} (a_k \prod_{m=1}^{k-1} b_m)$$

4 Various Notations of ALL Notation

We can show various series of numbers and operations with different forms of ALL notation. For instance, let:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_n \\ \star \end{bmatrix} = \underset{n=x}{\overset{y}{\star}} a_n$$

Therefore:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_n \\ + \end{bmatrix} = \underset{n=x}{\overset{y}{+}} a_n = \sum_{n=x}^{y} a_n$$

This is quite a simple notation that can clearly describe a series of values (a_n) being operated on by the binary operation $(\star)$. Recall that our ALL notation cannot denote something like a BCF (Branched Continued Fraction) in a closed form notation. Perhaps we can add a new form of the ALL notation. Recall the general BCF form is:

$$\underset{k=x}{\overset{y}{\mathrm{D}}} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)} = \sum_{i_x=r_x}^{N_x} \frac{c_{i_x}(z)}{d_{i_x}(z) + \underset{k=x+1}{\overset{y}{\mathrm{D}}} \sum_{i_k=r_k}^{N_k} \frac{c_{i_k}(z)}{d_{i_k}(z)}}$$

Trying to substitute ALL notation for the summation may lead to confusion. The functions c and d in the notation are of course separate arguments of this form of the branched continued fraction. Let:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_{1,n} & a_{2,n} & \cdots & a_{z,n} \\ \star_{1,n} & \star_{2,n} & \cdots & \star_{z,n} \\ \bigstar_1 & \bigstar_2 & \cdots & \bigstar_z \end{bmatrix}$$

Equate to:

$$(\underset{n=1}{\overset{\infty}{\bigstar_1}}(a_{1,x}\star_{1,x}(\underset{n=1}{\overset{\infty}{\bigstar_2}}(a_{2,x}\star_{2,x}(\cdots \underset{n=1}{\overset{\infty}{\bigstar_z}}(a_{z,x}\star_{z,x}(\underset{n=x+1}{\overset{y}{ALL}}\begin{bmatrix} a_{1,n} & b_{2,n} & \cdots & c_{z,n} \\ \star_{1,n} & \star_{2,n} & \cdots & \star_{z,n} \\ \bigstar_1 & \bigstar_2 & \cdots & \bigstar_z \end{bmatrix}))))))))$$

Therefore:

$$\underset{k=x}{\overset{y}{\mathrm{D}}} \sum_{i_k=1}^{\infty} \frac{c_{i_k}(z)}{d_{i_k}(z)} = \sum_{i_x=1}^{\infty} \frac{c_{i_x}(z)}{d_{i_x}(z) + \underset{k=x+1}{\overset{y}{\mathrm{D}}} \sum_{i_k=1}^{\infty} \frac{c_{i_k}(z)}{d_{i_k}(z)}}$$

Is equal to:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} c_n(z) & d_n(z) \\ \div & + \\ + & \bigstar_{\text{return first}} \end{bmatrix}$$

We would need even more arguments to properly define the bounds for each for-loop at the new bottom row of the array argument. Therefore, instead of just a binary operation input for a value in the bottom row, we could have the bottom row have table inputs, each with one value as a binary operation and two number values for the upper and lower bounds. Interestingly:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_{1,n} & a_{2,n} & \cdots & a_{z,n} \\ \star_1 & \star_2 & \cdots & \star_z \\ \bigstar_{\text{return first}} & \bigstar_{\text{return first}} & \cdots & \bigstar_{\text{return first}} \end{bmatrix} = \underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_{1,n} & a_{2,n} & \cdots & a_{z,n} \\ \star_1 & \star_2 & \cdots & \star_z \end{bmatrix}$$

It appears that this form of ALL notation can mostly describe the general BCF notation. We may be able to simply define new operations to redefine the new ALL notation. For instance, let:

$$a \star_f b = \sum(a+b)$$

Therefore:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_n \\ \star_f \end{bmatrix} = \sum(a_x + \sum(a_{x+1} + \sum(\cdots a_{y-2} + \sum(a_{y-1} + a_y))))$$

This is equivalent to:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} a_n \\ + \\ + \end{bmatrix}$$

However, can our new ALL notation describe:

$$x + \cfrac{}{x+1+\cfrac{x+2+\cfrac{x+3+\cfrac{x+4+\cfrac{x+5+\cdots}{x+6+\cdots}}{x+5+\cfrac{x+6+\cdots}{x+7+\cdots}}}{x+4+\cfrac{x+5+\cdots}{x+6+\cdots}}}{x+3+\cfrac{x+4+\cfrac{x+5+\cdots}{x+6+\cdots}}{x+5+\cfrac{x+6+\cdots}{x+7+\cdots}}}}{x+2+\cfrac{x+3+\cfrac{x+4+\cfrac{x+5+\cdots}{x+6+\cdots}}{x+5+\cfrac{x+6+\cdots}{x+7+\cdots}}}{x+4+\cfrac{x+5+\cfrac{x+6+\cdots}{x+7+\cdots}}{x+6+\cfrac{x+7+\cdots}{x+8+\cdots}}}}$$

Where the nth iteration of the value above adds 2^n numerators and denominators. Perhaps creating a new binary operation may simplify the difficulty presented by this problem. Although, perhaps we should add the rule that the binary operation for this scenario must have a closed form equality. This implies that infinite summation or products are invalid for the equality of the binary operation. However, the way the fraction expands above is very difficult to describe even with regular binary operations with finite expansions. We could also try to write the function below with ALL notation:

$$\text{DBCF}(x) = \cfrac{x}{\cfrac{x+1}{\ddots\,\cfrac{x+2}{x+2}} + \cfrac{x+2}{\ddots\,\cfrac{x+3}{x+3}} + \cfrac{x+3}{\ddots\,\cfrac{x+4}{x+4}} + \cfrac{x+4}{\ddots}}$$

However, like the other function, it branches out in a way that involves multiple binary operations for each branch. However, because it looks so similar to our BCF definition, we may be able to still turn it into ALL notation. Because there are branches that continue the pattern, we would need to use the third row of our array. However, because there are a finite amount of branches per iteration, we would have to add another argument for the upper bound of the operational series of the bottom row. Let:

$$\underset{n=x}{\overset{y}{ALL}} \begin{bmatrix} a_{1,n} & a_{2,n} & \cdots & a_{z,n} \\ \star_{1,n} & \star_{2,n} & \cdots & \star_{z,n} \\ \bigstar_1^u & \bigstar_2^u & \cdots & \bigstar_z^u \end{bmatrix}$$

Be equivalent to the original definition except the upper bounds of the operational series are the exponents. The operational series would also likely have to involve their indices in the equality to have their branches vary.

5 Order of Operations

In ALL notation, the binary operations in parentheses ALWAYS go first. Therefore, PEMDAS is unneeded for ALL notation unless the equalities for the binary operations in the notation have binary operations that do follow PEMDAS (like $(a \star (b)) = a + b \div a$). However, what if we made a new notation where:

$$\text{OperationalSequenceWithPEMDAS}(a_1, a_2, \cdots, a_n; \star_1, \star_2, \cdots, \star_{n-1})$$

Equals:

$$a_1 \star_1 a_2 \star_2 \cdots \star_{n-1} a_n$$

This of course is not that similar to ALL notation due to for-loops not being nested from the index to the upper bound because there are no indices or upper bounds in this notation. This notation also lacks parentheses, which of course may lead to interesting properties from PEMDAS. To know the order in which binary operations occur, we can add an integer index to each binary operation as an argument, where binary operations with the largest indices are done first. We can write the binary operation like $a \star_n [\text{INDEX}]b$, where the subscript n defines what the binary operation is whereas the INDEX in the brackets is an integer that shows the order of the binary operations relative to other binary operations. Therefore:

$$a \star_1 [5]b \star_2 [-2]c \star_3 [9]d$$

Is equivalent to:

$$((a \star_1 b) \star_2 (c \star_3 d))$$

For simplicity, starting the indices of a relation's binary operations from 1 or 0 is usually preferable than starting at any other integers. This system works

decently well, enabling you to reorder some of the parts of an expression around by changing the indices without changing the expression's value. However, we do not need PEMDAS, indices or parentheses if we simply write an expression left to right where the operations on the left occur first. For instance:

$$a^{bc+d} - e = b * c + d \uparrow^{-1} a - e$$

Such that:

$$a \uparrow^{-1} b = b^a = b \uparrow a$$

Because not all binary operations are commutative like addition and multiplication, we would likely use what we will call reversed binary operations. Given:

$$a \star_f b = f(a, b)$$

We get:

$$a\text{REVERSE}(\star_f)b = f(b, a)$$

Also:

$$f(a, b) = \text{REVERSE}(f)(b, a)$$

So instead of using an exponent of -1 on the binary operations that could be confused for inverse operations, we will use this REVERSE function for now. Perhaps to shorten the name of the function, we will abbreviate it to REV. Many also call this function the flip function. REV takes a single binary operation or binary function input and outputs another binary operation, where, of course, the new binary operation is the original binary operation with switched inputs. Composing the REV function with itself will return the original argument. Therefore:

$$\text{REV}(\text{REV}(\star)) = \star \implies \text{REV}(\star) = \text{REV}^{-1}(\star)$$

However, the fact that the REV function is an inverse to itself provides little help simplifying problems unless you encounter the function being nested. Back to our ALL notation, we can avoid the usage of parentheses altogether by using the functions and methods shown above. Of course, many do not even need parentheses, such as:

$$\overset{y}{\underset{n=x}{ALL}} \begin{bmatrix} f(n) \\ + \end{bmatrix}$$

$$\overset{y}{\underset{n=x}{ALL}} \begin{bmatrix} f(n) \\ * \end{bmatrix}$$

Where removing or adding all of the parentheses of the original formula on the notations above would not change the value. Furthermore, any commutative binary operation works. However, some binary operations that are NOT commutative also work. If we are operating from left to right, then:

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} f(n) \\ \star_{\text{return first}} \end{bmatrix}$$

Always returns the same value ($f(x)$) even without parentheses. The return first binary operation is definitely not commutative, but it is associative. Thus, many associative binary operations also work. What may be more interesting is if removing the parentheses from the ALL notation with two different binary operation inputs and top row inputs can return the same value. There are in fact many that satisfy this.

$$\underset{n=x}{\overset{y}{ALL}}\begin{bmatrix} f(n) & g(n) \\ \star_{\text{return first}} & \star \end{bmatrix}$$

Is simply one solution, operating from left to right. Overall, this section slightly described how PEMDAS can affect or not affect the ALL notation.

6 n-ary Operations

n-ary operations are operations or functions with n inputs where n is a natural number. The ALL notation takes binary operation inputs, but perhaps we can modify the notation to take operations with a different number of inputs. Unary operations are not reasonable in the ALL notation because they do not operate on more than one operand, making it impossible to convert the list of values in the top row of the matrix argument of the ALL notation into a single value with unary operations. However, if n is greater than 1, we could perhaps make the ALL notation work. We will try making the ALL notation compatible with ternary operations (3-ary operations). Recall:

$$\begin{bmatrix} \\ \star_f \left\{ \begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right. \\ \\ \end{bmatrix} = f(a_1, a_2, a_3)$$

However, it is preferable to write ternary operations as a function instead of using an operator. Recall:

$$\underset{n=1}{\overset{x}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_{f_1} & \star_{f_2} & \cdots & \star_{f_y} \end{bmatrix}$$

Is equal to:

$$f_1(a_{1,1}, f_2(a_{1,2}, \cdots f_y(a_{1,y}, f_1(a_{2,1}, f_2(a_{2,2}, \cdots f_y(a_{2,y}, \cdots f_{y-1}(a_{x,y-1}, a_{x,y}))))))))$$

However, all f_n for all n are binary functions in the already messy notation above. Perhaps the equality above would be more legible if compositions are used. However, we will have to define what composition we would need for these binary functions. Acknowledging the second argument is the argument being composed, we can get the notation:

$$(f \circ_2 g)(a, b) = f(a, g(b))$$

Where the subscript in $\circ_s$ composes the function g in the sth argument of the function f. A problem with this notation, however, is that we are unable to nest these functions multiple times without some minor adjustments since g appears to be a single-input function. Instead of this, Let:

$$(f_1 \circ_2 f_2 \circ_2 f_3)(a_1, a_2, a_3) = f_1(a_1, f_2(a_2, a_3))$$

Where the last function is ignored completely. This implies:

$$(f_1 \circ_2 f_2 \circ_2 f_3 \circ_2 \cdots \circ_2 f_n)(a_1, a_2, a_3, \cdots, a_n) = f_1(a_1, f_2(a_2, \cdots f_{n-1}(a_{n-1}, a_n)))$$

And furthermore, given:

$$g(x, y) = f_1(a_{x,1}, f_2(a_{x,2}, \cdots f_{z-1}(a_{x,z-1}, f_z(a_{x,z}, y))))$$

We get:

$$\overset{x}{\underset{n=1}{ALL}} \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,z} \\ \star_{f_1} & \star_{f_2} & \cdots & \star_{f_z} \end{bmatrix}$$

Is equal to:

$$g(1, g(2, g(2, \cdots (g(x, y)))))$$

Where $f_z(a_{x,z}, y) = a_{x,z}$. Although this notation above is not that great and $a_{x,z}$ is likely not in the domain of g for any argument y. Hence, the definition shown in the introduction on how ALL notation can be executed by a program is the best definition to apply ternary operations on. Now that we slightly understand ternary operations, we need a way to differentiate their symbols from binary operations in our new notation. So in the notation below, let f_n be a ternary function for all n. Let:

$$\overset{\infty}{\underset{n=1}{ALL}} \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_{f_1} & \star_{f_2} & \cdots & \star_{f_y} \end{bmatrix}$$

Equal:

$$f_1(a_{1,1}, a_{1,2}, f_2(a_{1,3}, a_{1,4}, \cdots))$$

Interestingly, our *All* notation from the introduction which represents the general form of ALL notation takes 5 inputs, making it a quinary operation of three table inputs and two integer inputs. Of course, you can combine all of the inputs into one large table if you wanted to.

7 Execution

Recall that our *All* function from the introduction had a clear algorithm to execute ALL notation. However, we can improve upon it. Let:

$$All_2(A,B,C,D) = \underset{n\in D}{ALL} \begin{bmatrix} C[1](A[1],n) & C[2](A[2],n) & \cdots & C[\#C](A[\#C],n) \\ \star_{B[1]} & \star_{B[2]} & \cdots & \star_{B[\#C]} \end{bmatrix}$$

Where the subscript of 2 shows this function is different from the last. $X[x]$ would be the xth value of list X. Having the index for the ALL notation be from a set allows much more flexibility. To execute this function in the Lua code, we get:

local function $All(A,B,C,D)$
local Output = 0
if $\#A==\#B$ and $\#B==\#C$ then
for i = $\#D$, 1, -1 do
for ii = $\#B$, 1, -1 do
Output = B[ii](C[ii](A[ii],D[i]), Output)
end
end
return Output
end
end

Or more accurately:

local function $All(A,B,C,D)$
local Output = 0
if $\#A==\#B$ and $\#B==\#C$ then
for i = $\#D$, 1, -1 do
for ii = $\#B$, 1, -1 do
if i == $\#D$ and ii == $\#B$ then
Output = C[ii](A[ii],D[i])
else
Output = B[ii](C[ii](A[ii],D[i]), Output)
end
end
end
return Output
end

end

This more generalized algorithm can execute almost all forms of ALL notation, especially if you add a fifth argument that also modifies the binary operations based on the index (like $\star_{\text{column, index}}$). You would simply have to verify that all of the binary operations are actually defined. The algorithm above is perhaps a simple genralization of the algorithm for reductions/folds shown on this video: `https://www.youtube.com/watch?v=d1peBFDPEUw`. If $\#A \neq \#B$ or $\#B \neq \#C$, then the function returns nothing (nil). It could also be modified to return an error message or 0 if they are unequal. The reason why the index is usually not a subscript of the binary operations is because the inputs for them are already affected by the index in the algorithm above. Therefore, you can modify the binary operations specifically for those values affected by the index if you wanted to. This is similar to how the third row in the matrix argument is not usually added because you can simply define your binary operations to have a series in them. There is a problem about this notation and algorithm we have not explored yet in this paper: what if the number of values and operations was infinite (the number of columns in the ALL notation's array is infinite)? We could simply substitute infinity (math.huge) for $\#A$, $\#B$, and $\#C$. Of course, if there are infinite binary operations or the notation has an infinite upper bound, a device could never completely compute the value with the algorithm above. Of course, the various operations and formulas can guide one usually to an exact, simplified answer. As you may have noticed, the for loops all start at the end of the ALL notation expression and work their way backwards by subtracting 1 from the indices when each loop is completed. One problem with the algorithm above is how we define the set D. The algorithm above has the index i equal to specific values of D in a specific order. This makes D (and all of the other tables) are not like a set because given the set:

$$X = Y \bigcup Z$$

We know that:

$$X = Z \bigcup Y$$

However, changing the order of D would change how the index changes in the ALL notation. Therefore, D can be considered a table or sequence like the rest of the variables. Therefore, even though the notation:

$$All_2(A, B, C, D) = \underset{n \in D}{ALL} \begin{bmatrix} C[1](A[1], n) & C[2](A[2], n) & \cdots & C[\#C](A[\#C], n) \\ \star_{B[1]} & \star_{B[2]} & \cdots & \star_{B[\#C]} \end{bmatrix}$$

Shows that $n \in D$ like D is a set, D is actually a table like the rest of the variables of the function and the input starts at D[1] and goes up to $D[\#D]$ in the algorithm. This is an extremely important concept because unlike sigma

notation or product notation, the order the index goes in matters because of the various operations. Furthermore, this implies that having the index of the ALL notation be an element of a set can generate multiple possibilities for its expanded expression. One way to provide less ambiguity for the notation above is by rewriting it as:

$$All_2(A,B,C,D) = \underset{n=1}{\overset{\#D}{ALL}} \begin{bmatrix} C[1](A[1],D[n]) & C[2](A[2],D[n]) & \cdots & C[\#C](A[\#C],D[n]) \\ \star_{B[1]} & \star_{B[2]} & \cdots & \star_{B[\#C]} \end{bmatrix}$$

Thus, the notation above is the most specific and accurate generalization of the ALL notation yet. Because of the problems with sets in the ALL notation, this paper will avoid setting the indices of future ALL notations to be elements of sets, unless all of the possibilities from the set are identical when simplified. To clearly explain what I mean, we know that:

$$\sum_{n \in \{x\} \bigcup \{y\}} = x + y = y + x$$

And

$$\prod_{n \in \{x\} \bigcup \{y\}} = x * y = y * x$$

However, this only works in the notations shown above because for these operations:

$$x \star y = y \star x$$

And

$$x \star (y \star z) = y \star (x \star z) = z \star (y \star x)$$

However, the likelihood of many operations behaving like the ones above can be quite low when dealing with multiple unique binary operation inputs in the ALL notation. Overall, here are the various datatypes in $All_2(A,B,C,D)$: A and D are lists of numbers; B and C are lists of binary functions. We can rewrite simple forms of ALL notation (specifically ALL notation with a single column in its argument matrix) into a right or left fold (foldr and foldl functions).

$$\underset{n=x}{\overset{y}{ALL}} \begin{bmatrix} a_n \\ \star_f \end{bmatrix} = f(a_x, f(a_{x+1}, \cdots f(a_{y-1}, a_y) \cdots))$$

Therefore

$$\underset{n=x}{\overset{y}{ALL}} \begin{bmatrix} a_n \\ \star_f \end{bmatrix} = \text{foldr } f \ a_y \ [a_x, a_{x+1}, \cdots, a_{y-1}]$$

And

$$\underset{n=x}{\overset{y}{ALL}} \begin{bmatrix} a_n \\ \star_f \end{bmatrix} = \text{foldl REV}(f) \ a_y \ [a_{y-1}, a_{y-2}, \cdots, a_x]$$

As you can see, init for both folds is a_y, which is an interesting property for folds. Although folds are how many of these sequences of operands with a single operator are described, adding more unique columns to the argument matrix of ALL notation complicates describing the fold much more. Thus, given a single-operational series, describing it with folds is probably better than using ALL notation. However, ALL notation is superior to folds when there are multiple operations and lists of operands, thus making it perfect for describing nearly all multi-operational series.

8 Total Operational ALL Notation Convergence

In this section, we will define the concept of total operational convergence and how we can see it in ALL notation. We know the ALL notation for a convergent infinite continued fraction:

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} A_n & B_n \\ \div & + \end{bmatrix} = \lim_{n\to\infty} \frac{A_1}{B_1 + \frac{A_2}{B_2 + \cdots \frac{A_n}{B_n}}} = c$$

Imagine if we added A_{n+1} to B_n:

$$\lim_{n\to\infty} \frac{A_1}{B_1 + \frac{A_2}{B_2 + \cdots \frac{A_n}{B_n + A_{n+1}}}}$$

If:

$$\lim_{n\to\infty} \frac{A_1}{B_1 + \frac{A_2}{B_2 + \cdots \frac{A_n}{B_n + A_{n+1}}}} = \lim_{n\to\infty} \frac{A_1}{B_1 + \frac{A_2}{B_2 + \cdots \frac{A_n}{B_n}}} = c$$

Then we can state the ALL notation for the original equation has total operational convergence. However, this is just one simple case of total operational convergence. For a more general case, given:

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix}$$

Which equals:

$$\lim_{x\to\infty} a_{1,1}\star_1(a_{1,2}\star_2(\cdots a_{1,y-1}\star_{y-1}(a_{1,y}\star_y(a_{2,1}\star_1(a_{2,2}\star_2(\cdots a_{2,y-1}\star_{y-1}(a_{2,y}\star_y(\cdots a_{x,1}\star_1(a_{x,2}\star_2(\cdots a_{x,y-1}\star_{y-1}a_{x,y}))))))))))$$

And converges to some value c, then if:

$$\lim_{x\to\infty} a_{1,1}\star_1(a_{1,2}\star_2(\cdots a_{1,y-1}\star_{y-1}(a_{1,y}\star_y(a_{2,1}\star_1(a_{2,2}\star_2(\cdots a_{2,y-1}\star_{y-1}(a_{2,y}\star_y(\cdots a_{x,1}\star_1(a_{x,2}\star_2(\cdots a_{x,y-1}\star_{y-1}a_{x,y}\star_y$$

$$\underset{n=x+1}{\overset{x+1}{(ALL(z)}}\begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix}))))))))))$$

Equals c for all z where $z \in [1, y]$ and $z \in \mathbb{Z}$ and:

$$\underset{n=x}{\overset{x}{ALL}}(z) \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix} = a_{x,1} \star_1 (a_{x,2} \star_2 \cdots (a_{x,z-1} \star_z a_{x,z})) = \underset{n=x}{\overset{x}{ALL}} \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,z} \\ \star_1 & \star_2 & \cdots & \star_z \end{bmatrix}$$

Then the original operational series is totally operationally convergent. Notice how the new z argument transforms the $y \times 2$ matrix to a $z \times 2$ matrix. Of course, if $a_{x,z}$ or $\star_z$ are not defined in the original ALL notation, then it is undefined. Thus:

$$\underset{n=x}{\overset{y}{ALL}}(z) \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,y} \\ \star_1 & \star_2 & \cdots & \star_y \end{bmatrix} = \underset{n=x}{\overset{y}{ALL}} \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,z} \\ \star_1 & \star_2 & \cdots & \star_z \end{bmatrix}$$

All infinite operational series with a 1×2 matrix that converge are totally operationally convergent. Therefore:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} \frac{1}{n^2} \\ + \end{bmatrix} \text{ is totally operationally convergent}$$

While

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} \frac{1}{n} \\ + \end{bmatrix}$$

Is not. For the case where there are two rows and two columns, we will return to CFs.

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} 1 & 1 \\ \div & + \end{bmatrix} \text{ is totally operationally convergent}$$

While

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} 1 & -1 \\ \div & + \end{bmatrix} \text{ is not totally operationally convergent}$$

This is because $\frac{1}{-1} = -1$ and ending with a -1 in the denominator approaches $-1/\phi$ while adding 1 in the denominator makes the value undefined. If the infinite continued fractions or any infinite operational series diverges in general, then it is automatically not totally operationally convergent. This topic can be quite interesting for ALL notation because it usually involves many inputs of values and operations.

9 Operations on Operations in ALL Notation

We know that ALL notation applies a list of operations on a list of various values in a specific order. These values in the top row of the argument matrix for the notation are not necessarily numbers. They could be Boolean values, sets, lists,

even operations. However, what is an operation on an operator? Furthermore, how can we apply this to ALL notation? Usually applying an operation to an operator changes the operator such that it operates on values differently than if originally did. A classic example is the composition of two functions, returning a new function.

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} \star_f \\ \circ \end{bmatrix} = f \circ f \circ f \cdots = \underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} f \\ \circ \end{bmatrix}$$

We can also define new operations that specifically take operations as inputs. However, the main operation we will be analyzing in this section is the composition operation. The notation above works because an operator $\star_f$ without any operands is equivalent to the function f without any inputs in this context. No parentheses for the compositions are required because there is only a single function that is used for the operands for the compositions. However, the function f must take only 1 input based on the definition of the composition operator. In this section, we will demonstrate how we can use ALL notation to infinitely or finitely compose 1 or more functions, and we will visualize its results. Given:

$$f(x) = x$$

Then:

$$(\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} f \\ \circ \end{bmatrix})(x) = f(x) = x$$

However, if $f(x) = x + n$ where $n > 0$, then:

$$(\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} f \\ \circ \end{bmatrix})(x) \text{ diverges towards positive infinity}$$

This shows how a minor addition to a function can cause it to diverge. For a more advanced function such as $f(x) = \sin(x)$, we can find many interesting properties if we infinitely compose said function.

$$\sin(\sin(\sin(\cdots \sin(x) \cdots))) = 0$$

Here are some more examples of infinite compositions:

$$g(g(g(\cdots g(x) \cdots))) = \begin{cases} 0, -1 < x < 1 \\ 1, x = 1 \\ -1, x = -1 \text{ and } n \text{ is odd} \\ \text{diverges}, x > 1 \text{ or } x < -1 \end{cases}$$

Where $g(x) = x^n, n > 1$

$$h(h(h(\cdots h(e^x) \cdots))) = e^x, h = \frac{d}{dx}$$

For a more general problem, given:

$$j(j(j(\cdots j(x)\cdots))) = a$$

We know that:

$$j(a) = a$$

Implying for many functions that a is likely one of these 3 values: $\pm 1, \pm\infty, 0$. The function j may also simply diverge after being infinitely composed with some input x. Perhaps just as intriguing:

$$x = j^{-1}(j^{-1}(j^{-1}(\cdots j^{-1}(a)\cdots)))$$

However, there may not be a proper inverse for the function j for the equation above to work.

10 Solving for Arguments of Operational Series

In this section, instead of simplifying ALL notation, we will be solving for arguments in the ALL notation, including the top row matrix values, the binary operations, and the upper/lower bounds of the ALL notation. However, many of these solutions may not just be one value or operation. Perhaps the simplest ALL notation to solve for arguments are ones with a single column in their matrix argument. For instance:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a_n \\ + \end{bmatrix} = 1$$

There are many possible solutions to a_n, such as $a_n = \frac{1}{2^n}$ or $a_n = -n + 1.5 + |n - 1.5|$. However, because the first solution listed is continuous and gradually converges, so it is probably preferable to the other solution. There are many more solutions for a_n, but how could we solve:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} a \\ + \end{bmatrix} = 1$$

There is no solution for a if a is a constant. However:

$$\lim_{x\to\infty} \underset{n=1}{\overset{x}{ALL}} \begin{bmatrix} a \\ + \end{bmatrix} = 1, a = \frac{1}{x}$$

Instead of solving for a value in the top row of the matrix, we can solve for a binary operation is the bottom row. Given:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} 1 \\ \star_f \end{bmatrix} = 1$$

Where $a \star_f b = f(a, b)$, we have numerous solutions for $\star_f$. Some of these solutions for f are $f(a, b) = a$, $f(a, b) = b$, and $f(a, b) = ab$. We can also solve for the bounds or set of inputs for the index. Given:

$$\underset{n=1}{\overset{x}{ALL}} \begin{bmatrix} 1 \\ + \end{bmatrix} = 100$$

We know $x = 100$. In many situations however, there may be no solution for the upper or lower bounds. When we add another column to the argument matrix, solving for arguments can be much more difficult. For instance:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} 1 & x \\ \div & + \end{bmatrix} = 1$$

Can be quite difficult to solve for x. Because this is an infinite ALL notation, we can simplify this to:

$$1 \div (x + (1)) = 1$$

Solving for x, we get $x = 0$. However, if $x = 0$ and this always ends in $1 \div x$, this would actually be undefined if $x = 0$. Therefore, there is no solution for x. Thus, solving for a value is not as simple as substituting what the ALL notation is equal to. However, it may work in some cases:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} x & 1 \\ \star_{\text{return first}} & + \end{bmatrix} = 1$$

$$\implies x \star_{\text{return first}} (1 + (1)) = 1$$

$$\implies x = 1$$

In this scenario, it is true that $x = 1$ after solving the equality from substituting the equality of the ALL notation in for where it repeats. However, you must check all domain restrictions and input the values back into the ALL notation to verify whatever solutions you get are true. When there are finite upper and lower bounds for ALL notation, solving for a value can be easier than infinite with bounds. This is because finite bounds would imply a finitely long expression (unless the binary operations used in the ALL notation create an infinitely long expression when simplified). What may be more interesting is solving for possible binary operations that could satisfy a relation involving ALL notation. For instance, we can try to solve for the binary function f in:

$$\underset{n=1}{\overset{\infty}{ALL}} \begin{bmatrix} 1 \\ \star_f \end{bmatrix} = e, a \star_f b = f(a, b)$$

In this scenario, we could get a few solutions for f such as $f(a, b) = e$. To find more solutions, we must acknowledge that:

$$f(1, e) = e$$

$$f(1, f(1, f(\cdots 1, f(1, 1) \cdots))) = e$$

If the function abides by the rules above while not failing any domain restrictions, then it is valid. We can find solutions $f(a, b) = abe$ and $f(a, b) = e + a - 1$.

However, these solutions may be considered trivial because they directly involve the solution to the ALL notation within them (e). Finding a nontrivial solution for this problem can prove to be quite challenging. One possible solution is $f(a,b) = \lim_{x\to\infty}(a+\frac{a}{x})^x$. However, this limit definition basically is e already. Furthermore, nontrivial solutions should gradually converge to the equality of the ALL notation through more compositions of f. With all of these rules, there is no simple solution for e in this scenario because that would imply that e can be equated to a finite expression of standard binary operations and 1s. Although, trivial cases are still solutions, so, all that needs to be satisfied for the solutions of a more general case:

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} x \\ \star_f \end{bmatrix} = y, a \star_f b = f(a,b)$$

Are:

$$f(x,y) = y$$
$$f(x, f(x, f(\cdots x, f(x,x)\cdots))) = y$$

When we add more binary operations, solving for one can yield interesting results. Given:

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} 1 & 1 \\ \div & \star_f \end{bmatrix} = x, a \star_f b = f(a,b)$$

One possible solution may be $f(a,b) = 1/x$. This solution is valid because:

$$\frac{1}{f(1, \frac{1}{\cdots f(1,1)})} = x$$

However, this solution does not necessarily work with our previous restriction that states:

$$f(a,x) = x$$

Where, in actuality:

$$f(1,x) = \frac{1}{x}$$

This of course is because there are multiple binary operations, so we must compensate for them. If we add a third argument n as a subscript of the function in:

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} 1 & 1 \\ \div & \star_{f_n} \end{bmatrix} = x, a \star_{f_n} b = f_n(a,b)$$

Then we can easily get a piecewise function solution:

$$f_n(a,b) = \begin{cases} \frac{1}{x}, n = 1 \\ 1, n \neq 1 \end{cases}$$

Thus, solving for variables/functions in ALL notation can be a little challenging, but solutions can reveal the unique beauty of operational series.

11 A Continuous Notation

As you may have noticed, the ALL notation we have been using is discrete, not continuous, as summation is discrete to integration. As shown on this Wikipedia page:

`https://en.wikipedia.org/wiki/Product_integral`

Integration (continuous summation) can be represented by the the following formula:

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum f(x_i)\Delta x$$

This clearly shows how to describe continuous summation with discrete summation. However, you can do the same thing with product notation (not completely accurate):

$$\prod_a^b (1 + f(x)dx) = \lim_{\Delta x \to 0} \prod (1 + f(x_i)\Delta x)$$

Or more accurately:

$$\prod_a^b f(x)^{dx} = \lim_{\Delta x \to 0} \prod f(x_i)^{\Delta x}$$

This shows how you can turn the discrete form of product notation into a continuous form. We can generalize this using ALL notation.

$$\underset{a}{\overset{b}{ALL}} \begin{bmatrix} f(x)dx \\ \star \end{bmatrix} = \lim_{\Delta x \to 0} ALL \begin{bmatrix} f(x_i)\Delta x \\ \star \end{bmatrix}$$

Furthermore, a continuous ALL notation is created when you take the step size of the index for a normal ALL notation (likely 1) and you lower it (increasing the number of terms) until it approaches 0 to get the continuous form of ALL notation. Because this is mostly involving the index with respect to one value x, we may be able to create continuous forms of ALL notation with multiple columns in their matrix argument. Perhaps a more basic equation to explain this more clearly would be:

$$\int_a^b f(x)dx = \lim_{n \to \infty} \sum_{k=1}^{n} f(a + \frac{b-a}{n}k)\frac{b-a}{n} = \lim_{\Delta x \to 0} \sum f(x_i)\Delta x$$

And thus:

$$\prod_a^b f(x)^{dx} = \lim_{\Delta x \to 0} \prod f(x_i)^{\Delta x}$$

Where $x_i = a + \frac{b-a}{n}k$ and $\Delta x = \frac{b-a}{n}$. But why is the dx an exponent in this notation one might wonder. It is because we are multiplying the intervals instead of adding them, so instead of multiplying $f(x)$ by dx like in integration, we exponentiate $f(x)$ to dx. This implies that how we defined the continuous ALL notation generalization is incorrect. Instead of multiplying by dx for all binary operation inputs, we must somehow incorporate the binary operation into the expression with dx. We know:

$$a \times b = \sum_{n=1}^{b} a$$

$$a^b = \prod_{n=1}^{b} a$$

Where b is a natural number. Due to the complexity of trying to combine the addition of areas with some other binary operation being basically impossible in some situations, a general continuous form of ALL notation likely could not exist for all inputs.

12 Uses for ALL Notation

If one word could sum up all of ALL notation, it would be the word clarity. When one encounters an ellipsis for some sequence of many repeated operations and operands, figuring out what goes next can be challenging. With ALL notation, you can clearly and compactly define any operational series. As we have seen previously, ALL notation has a straightforward algorithm that can clearly define any standard operational series. However, the uses of ALL notation do not end there. We can find the connections between binary operations and their operands, revealing a deeper understanding of them. The algorithm for ALL notation can be used to calculate various folds and reductions in many programming languages, which can be vital in some algorithms. In algorithms that use reductions and folds, comments showing the ALL notation for them can make readers understand the algorithms more.

13 Conclusion

ALL notation is an elegant way to represent any (multi-)operational series.

14 Resources

Below are some of the resources used for this paper:

`https://en.wikipedia.org/wiki/Binary_operation`
`https://en.wikipedia.org/wiki/Binary_function`
`https://www.wolframalpha.com/`

https://en.wikipedia.org/wiki/Product_integral
https://www.youtube.com/watch?v=d1peBFDPEUw
https://medium.com/@zaid.naom/exploring-folds-a-powerful-pattern-of-functional-programmi
https://en.wikipedia.org/wiki/Fold_(higher-order_function)
https://m.youtube.com/watch?v=WKKkIGncRn8

15 Extra

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}1 & 1 & 1\\ + & \div & +\end{bmatrix} = \sqrt{2} = 1 + \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}1 & 2\\ \div & +\end{bmatrix}$$

$$\overset{x-1}{\underset{n=1}{ALL}}\begin{bmatrix}1 & 1 & 1\\ + & \div & \div\end{bmatrix} = x = \overset{x}{\underset{n=1}{ALL}}\begin{bmatrix}1\\ +\end{bmatrix}, x \in \mathbb{N}$$

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}n & n\\ \div & +\end{bmatrix} = \frac{1}{e-1}$$

$$\overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}n & n\\ + & \div\end{bmatrix} = \frac{1}{e-2} \approx 1.3922$$

$$e = 2 + \overset{\infty}{\underset{n=2}{ALL}}\begin{bmatrix}n & n\\ \div & +\end{bmatrix}$$

$$\phi = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}1 & 1\\ + & \div\end{bmatrix} = 1 + \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}1 & 1\\ \div & +\end{bmatrix}$$

$$\phi = \overset{\infty}{\underset{n=1}{ALL}}\begin{bmatrix}1\\ \star_f\end{bmatrix}, a \star_f b = \sqrt{a+b}$$

$$All(A,B,C,D) = \underset{n\in D}{ALL}\begin{bmatrix}C[1](A[1],n) & C[2](A[2],n) & \cdots & C[\#C](A[\#C],n)\\ \star_{B[1]} & \star_{B[2]} & \cdots & \star_{B[\#C]}\end{bmatrix}$$

$$\lim_{x\to a}(\overset{y}{\underset{n=1}{ALL}}\begin{bmatrix}f_n\\ \circ\end{bmatrix})(x) = \lim_{x\to \lim\limits_{x_2\to a}(\overset{y}{\underset{n=2}{ALL}}\begin{bmatrix}f_n\\ \circ\end{bmatrix})(x_2)} f_1(x), f_1 \text{ is continuous}$$

$$\overset{k}{\underset{n=1}{ALL}}\begin{bmatrix}a & a\\ \star & \star\end{bmatrix} = \overset{2k}{\underset{n=1}{ALL}}\begin{bmatrix}a\\ \star\end{bmatrix}$$

$$\overset{k}{\underset{n=1}{ALL}}\begin{bmatrix}a & b\\ \star & \star\end{bmatrix} = \overset{2k}{\underset{n=1}{ALL}}\begin{bmatrix}\frac{a\left((-1)^{1+n}+1\right)+b((-1)^n+1)}{2}\\ \star\end{bmatrix}$$

$$\overset{k}{\underset{n=1}{ALL}}\begin{bmatrix}a_1 & a_2 & \cdots & a_x\\ \star & \star & \cdots & \star\end{bmatrix} = \overset{xk}{\underset{n=1}{ALL}}\begin{bmatrix}a_{\mathrm{mod}(n-1,x)+1}\\ \star\end{bmatrix} = \overset{xk}{\underset{n=1}{ALL}}\begin{bmatrix}a_{((n-1)\%x)+1}\\ \star\end{bmatrix}, x,k \in \mathbb{N}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a_n & b_n & c_n \\ \star_1 & \star_{\text{return first}} & \star_2 \end{bmatrix} = a_1 \star_1 b_1$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a_n & b_n & c_n \\ \star_1 & \star_{\text{return second}} & \star_2 \end{bmatrix} = \underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a_n & c_n \\ \star_1 & \star_2 \end{bmatrix}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a & 0 \\ \div & \star \end{bmatrix} = \text{ undefined}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a & b \\ \star & \star \end{bmatrix} = \underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a \star b \\ \star \end{bmatrix}, \star \text{ is commutative and associative}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a & b \\ \star & \star \end{bmatrix} = \underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} b & a \\ \star & \star \end{bmatrix}, \star \text{ is commutative and associative}$$

$$\text{mean}(A) = \underset{n=1}{\overset{\#A}{ALL}}\begin{bmatrix} \frac{A[n]}{\#A} \\ + \end{bmatrix}, A \text{ is a list of numbers}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a_n \\ \text{REV}(\star_f) \end{bmatrix} = f(f(\cdots f(a_k, a_{k-1}), a_{k-2} \cdots), a_1)$$

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} f \\ \star_f \end{bmatrix} = f(f, f(f, f(\cdots)))$$

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} \phi^{\frac{|n-2|-(n-2)}{2}} & x^{(-1)^n} \\ \div & + \end{bmatrix} = x, x \in \mathbb{C}$$

$$\underset{n=a}{\overset{b}{ALL}}\begin{bmatrix} c \\ \star \end{bmatrix} = 0, a > b$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a \\ \% \end{bmatrix} = \text{ undefined}, k > 2$$

$$\underset{n=1}{\overset{b}{ALL}}\begin{bmatrix} a \\ \uparrow^c \end{bmatrix} = a \uparrow^{c+1} b$$

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} a \\ \star \end{bmatrix} = \text{some constant } b, a \star z = \text{some constant } c, \forall z, a, c \in \mathbb{C}$$

$$\underset{n=1}{\overset{b}{ALL}}\begin{bmatrix} \vec{v} \\ \times \end{bmatrix} = 0, b > 1$$

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} a & b \\ \star_f & \star_f \end{bmatrix} = \text{undefined}, x \star_f y = \log_x(y)$$

$$\underset{n=1}{\overset{\infty}{ALL}}\begin{bmatrix} a & b \\ \star_f & \star_f \end{bmatrix} = 1, b > 0, a \in \mathbb{R}, x \star_f y = y^{\frac{1}{x}}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a \\ - \end{bmatrix} = \begin{cases} a, k \text{ is odd} \\ 0, k \text{ is even} \end{cases}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} A \\ \setminus \end{bmatrix} = \begin{cases} A, k \text{ is odd} \\ \varnothing, k \text{ is even} \end{cases}$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} f \\ \circ \end{bmatrix}(x) = \begin{cases} -x, k \text{ is odd} \\ x, k \text{ is even} \end{cases}, f(x) = -x \ \forall x$$

$$\underset{n=1}{\overset{k}{ALL}}\begin{bmatrix} a_n & b_n \\ \div & + \end{bmatrix} = \frac{A_k}{B_k}, k>0, A_x = b_x A_{x-1} + a_x A_{x-2}, B_x = b_x B_{x-1} + a_x B_{x-2}, A_{-1}=1, A_0=0, B_{-1}=0, B_0=1$$