Aarex's Oddly Strong Notation

Side-notation made by Aarex

Aarex's Superstrong Array Notation | Aarex's Zenith Strong Notation **(NEW!)** | Aarex's Googology

Aarex's Oddly Strong Notation (aOSN) is a side-notation made by Aarex, **which takes a different direction from aSAN.**

os(n) represents this notation, like ss(n) from aSAN.

At the beginning, all aSAN- rules are applied to 2-entry arrays. At 3 entry arrays, this is where the fun begins.

A different direction...

For aOSN, the process only compares the position of N's entry and its next one. Oh, the following entries must be equal.

However, this causes some infinite loop problems.

1. In (1,(((1,1,2),1,2),1,2)), (1,1,2) is the first entry of ((1,1,2),1,2). So, the process only compares the first and second entries. Because 1 is the second entry of N, it expands from the whole array. So, there is no supremum of a -> (((((1,a)),1,2)),1,2).
2. In (1,1,(1,2)), (1,2) is the third entry of (1,1,(1,2)). So, it only compares the third and fourth entries, but these entries of N are 1...

I won't forget!

To fix this, we need exception arrays. When the process finds an exception array, it compares normally.

There are 2 kinds of exception arrays:

(uppercase variables = arrays, lowercase variables = entries)

1. (**T**,(**T**,z,**x**,A),**y**,B) where **T** is trailing 1s and **x** > **y**.
   The parent becomes an exception.
   (like (1,1,(1,1,3,5),2))
2. (**T**,1,(**A**,B),C), where **T** is trailing 1s, and **A** has the same amount of entries as **T** but greater than **T**.
   The children become an exception.
   (like (1,1,(2,4,3,(6,3),...))

Formalizing...

(Check out the definition and comparison of arrays at the process document)

Rules 1 - 3 haven't been changed, and so the definition.

1. os(b,1) = b
   Base Rule
   a. os(b) = b
2. os(b,A) = os(b+1,pre(A)) (= os(b+1,A-1))
   Successor Rule
   a. Only applies when A is a successor array.
3. Standardize os(b,A) and A before beginning the process.
   Simplification
   a. Now, let A be the second entry of os(_).

Step 4 needs to be changed to include positioning.

4. Find the leftmost entry that is a successor array.
   Leftmost Process
   a. Let L_0 = A, p = 1, and t = 0.
   b. Let e = the p^{th} entry of L_t.
   c. If the entry (e) is 1, then move to the next entry.
      i. Increase p by 1.
      ii. Go back to Substep 4b.
   d. If the entry (e) is a Limit Array, then jump into it.
      i. Increase t by 1.
      ii. Let L_t = e **and P_t = p.**
      iii. Change p to 1.
      iv. Go back to Substep 4b.
   e. If the entry (e) is a Successor Array, then let N = L_t.

Determining exception arrays is essential for aOSN. Like determining Transcenders from aSAN, this is step 5 of the process.

5. **Find exception arrays for all layers of N.**
   Exception Barrier
   a. **If t = 0** (or P_t doesn't exist)**, then skip to step 7.**
      i. **Also, let B = L_0.**
   b. **Let k = 1.**
   c. **If P_k+1-th entry of L_k > P_k+1-th entry of L_{k-1}, then L_{k-1} is an exception array.**
      (treat nonexistent entries as all 1s)
   d. **If P_k > 2 and the first P_k-2 entries of L_k > (1,1...1,1), then L_k is an exception array.**
   e. **If k < t, then:**
      i. **Increase k by 1.**
      ii. **Go back to Substep 5c.**

Only Substep 6d has been splitted into 2 Substeps for the purpose of this notation.

Due to no aSAN rules, some Substeps have been omitted or simplified.

a. Let k = t.

b. If k = 0 or P_t doesn't exist, then:
    a. Let B = L_0.
    b. Jump to Step 7.
c. Decrease k by 1.
d. If **L_k is an exception array** and L_k < N, then
    a. Let B = L_{k+1}.
    b. Jump to Step 7.
e. **If L_k isn't an exception array,**
    entries of L_k after P_t+1th = entries of L_{k-1} after P_t+1th, and
    (P_t+1th entry of L_k < P_t+1th entry of N or
    P_tth entry of L_k < P_tth entry of N),
    (treat nonexistent entries as all 1s)

    then:
    a. Let B = L_{k+1}.
    b. Jump to Step 7.
f. Go back to Substep 6b.

The following definition hasn't been changed. Let's overview rule 7 before moving on.

7. Finally, we expand from B.
Ascender Expansion
    a. Let N(n) = N, but:
        i. Change S to pre(S). (= S-1)
        ii. Change the entry of N preceding S to n.
    b. Let B(n) = B, but change N to N(n).
    c. Change B to $B^b(b)$.
    d. The process ends.

Examples

Let's look at ((1,1,2)). Because (1,1,2) is the first entry, the process compares the first and second entries.

Due to these entries of N being 1, it expands from the whole array.

But with exception 1, it expands from (1,x) arrays instead.

(because (1,1,2) > (1,x))

One example is (1,(((1,1,2)),1,2)) = nests of a -> (1,((a),1,2)).

(Exception 1 makes ((1,1,2)) stops at the first layer)

Now, take (1,(1,1,2)). Because (1,1,2) is the second entry, the process compares the second and third entries.

(1,1,2) stops at this example, because the third entry is less than 2.

Huh?

There are several examples that significantly contribute to strength. Let's examine all of them.

Weaker

1 - (1,(((1,1,2)),1,2))

(1,_) layer is an exception array, because the third entry of the children is greater than 1.

The exception array is less than N, so N expands from that array.

2a - ((1,2,2))

((1,3,2),2) / (1,((1,3,2),2))

N stops at the parent, because the second entry of N is greater than the second entry of the parent.

2b - (((1,2,2),2))

(((1,2,2),2),1,2)

(((1,2,2),2)) is an exception array, but ((1,2,2),2) isn't. N stops at (1,1,2) because it is less than N.

Not a standard example due to the previous example.

3 - ((1,(1,1,2),2))

(1,1,2) is the second entry. It goes further than the parent (greater than N), but stops at the whole example which is less than N.

4 - (1,1,((1,1,2)))

Because the first entry is greater than 1 in the children of (1,1,_), (1,1,2) expands from that (x) array. The result stays the same unless you increase the third entry in (x) array.

That's the purpose of Exception 2.

5 - ((1,((1,2,2),2)),1,2)

This is one you need to watch out for. There is an array in the first entry, where the second entry of that is 2. (>1)

Therefore, the whole example is an exception. To clarify, N stops at (1,1,2) because it is less than N.

Stronger

1 - (1,((1,1,2)))

(((1,((1,1,2))),2),1,2) / ((1,((1,1,2))),1,2) / (((1,1,2),2),1,2)

This time, (1,1,2) is the first entry. Because (1,1) is the lowest array being compared, N expands from the whole array.

Also, (1,((1,1,2))) isn't an exception array because the third entry of the second entry is 1.

2 - ((1,2),1,2)

(((((1,2),3,2),1,2),1),3) / (1,((1,2),1,2))

N = (1,2) doesn’t stop at (1,1,2) because ((x,2),1,2) is an exception array.

3 - ((1,2,2),2)

(1,((1,2,2),2))

((1,2,2),2) is not an exception array because the second entries in that statement are both 2.

4 - ((1,1,3),1,2)

Because (1,1) is the lowest array being compared, N expands from the whole array. Just like the first example.

5 - (1,(1,(1,2),2))

(1,2) is the second entry, so the process compares the second and third entry.

Because (2,1) is below ((1,2),2), the only way to stop N is to put this example into (_,1,1,A) arrays.

6 - (1,1,(1,2))

(((1,1,(1,2))),1,(1,2)) / (1,(1,1,(1,2))) / (1,(1,1,(1,2)),(1,2)) / (1,1,((1,1,(1,2)),1,(1,2))

(1,2) is the third entry of (1,1,(1,2)), so the process only compares the third and fourth entry.

Because (1,1) is the lowest array being compared, N expands from the whole array. Not even (x,y,A) arrays stop it, except (1,1,(x)).

7 - (1,(1,1,1,2))

((1,(1,1,1,2))) / (1,1,(1,(1,1,1,2)))

(1,1,1,2) is the second entry of (1,(1,1,1,2)). But the second and third entries are 1. Therefore, N expands from the whole array.

Also, (1,(1,1,1,2)) isn't an exception array, because the first entries are 1. So exception 2 doesn't apply!

Continuing at this point...

(as of 10/25/21)

This notation is relatively new. So, the growth rate of this notation is currently unknown.

Since it might be one of the strongest notations, I won’t go further for now. See you later.

~ TehAarex, **Lightning**

(p.s. Higher Hypercascading Hyper Notation is not well-defined. I think blocks aren’t formally defined, but linear arrays are.

Same goes to hypercascaders, H-functions, and [X^^2X] stuff.
I saw it recently when I visited Nish’s new googology site!)

"Yes! I made it!" ~ Lightning, TPOT 1

Analysis

Compared with aSAN is really simple with this proof:

In ((1,1,2)), the second entry of both layers is 2.
Exception 1 applies to (1,(1,1,2)), because the third entry of the children is greater than the parent's one.
Therefore, the only way to stop ((1,1,2)) is to put this into (1,(_,1,2)).

(1,(1,2)) is a Transcender, which only looks for themselves. The only way to stop this is to put this into (1,(_)) arrays, which are also Transcenders.

Due to Stronger Example 1, (1,((1,1,2))) also works like ((1,1,2)) because it is the first entry, and the second entry of N is 1.
Also, in ((1,1,2)), the third entry is 1, so Exception 1 doesn't apply to that example.

In conclusion, ((1,1,2),n) might work like (1,n) in (1,_).

Binary (aOBN)

Aarex's Oddly Strong Binary Notation

Exception 2 can be ignored, because there is no 0th entry, even in children of the second entry.

Substep 8c can be ignored too. Here's a little proof why.
Exception 1 is applied to ((a,b),c) where b > c.
Limited comparison can't look at the third entry because all arrays must have at most 2 entries.
If N is the first entry, then the limited comparison only looks at the second entry.
However, exception 1 can take care of that.

However, for (((1,2),1,2)), the limited comparison must be applied to resolve infinite loops.

Other Problems

There are 2 problems whereas one condition is removed.

1. No Equal Condition:
In (((1,2),1,2),1,2), (1,2) is the first entry...
Because the second entry of the parent is less than 2, N would expand from that without the equal condition.
This type is a weakness.
(((1,2),1,2) is skipped due to being an exception array.)

2. 1-Entry Comparison:
In (((1,(1,2)),1,2)) (where N is the second entry)...
N would go further without the second entry comparison.
This type is infinite loops.

The 2-1 Theory

[Not proven yet]

In this theory, if you put an array with A_z into the kth entry of any array that works like (1,(1,1,3)), where..

- A_z = (1,1...1,1,n,B), with z-1 1s at the beginning, and z = k-1 or z >= k+2.

Then the whole array doesn't seem to change how it works! It is just the same thing!

Here's proof:

1. For A_{k+2}, A_{k+3}, ... in the kth entry, the parent doesn't match with exception 1 arrays.
2. For A_{k-1} in the kth entry, exception 2 doesn't apply because the entries before that are all 1!
3. However, N is less than the parent, which means N goes further!
4. During step 6, the process only compares k^{th} and $k+1^{th}$ entry, and the part that the process compares from N is (1,1)!

Numbers

aOSN numbers use the same naming system as aSAN, but adding "Oddly" on the left.

For example, Oddly Tetrke Entran = os(100,1,1,1,2)

However, the Finale group has different suffixes.

- **Lightning** Tractus = os(100,1,1...1,1,2) w/ 99 1s
  Also called Oddly Lightning Enus
- **Lightning** Oddius = os(100,100,100...100,100,100) w/ 101 100s
  Also called Oddly Lightning Ulus
- **Lightning** Escenus = os(100,1,1...1,1,(1,1...1,1,2)) w/ 99 1s for every trailing 1s
  Also called Oddly Lightning Trans, just behind S(1,919)!

Here's one more irregular number, and the legendary one that I won't forget...

Lightning Ziggarius = os(100,N_1)

$N_1 = (N_2)$

$N_2 = (1,1,N_3)$

$N_3 = (1,N_4)$

$N_4 = (1,1,1,N_5)$

$N_5 = (1,1,1,1,N_6)$

$N_6 = (1,1,1,1,1,1,N_7)$

$N_7 = (1,1,1,1,1,N_8)$

$N_8 = (1,1,1,1,1,1,1,N_9)$

...

$N_{101} = 2$

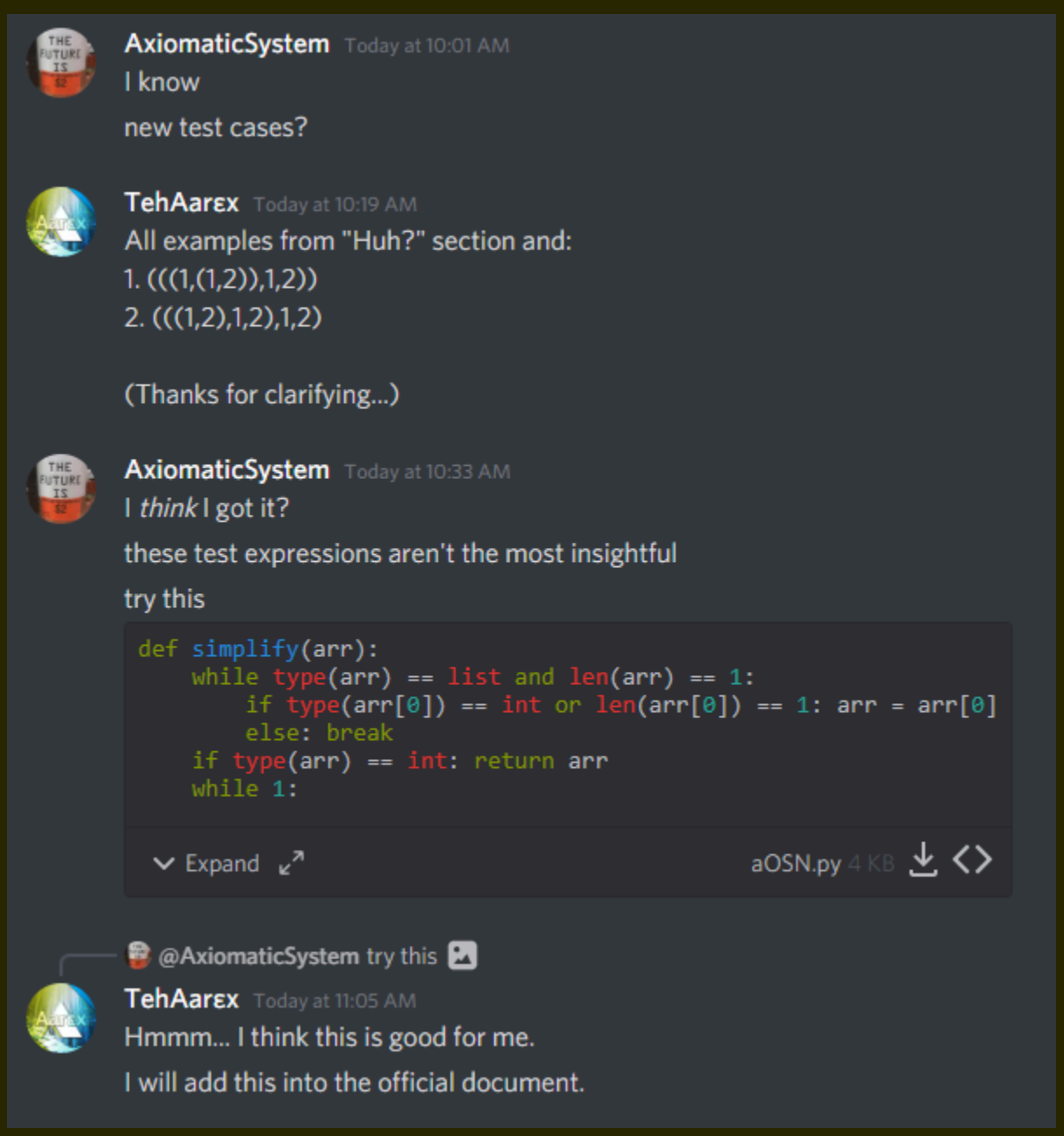
This pattern can be generalized:

- M(n,m) = (1,1,1...1,1,1,m) w/ n 1s
  - $N_{4x+4} = M(4x-1,N_{4x+1})$
  - $N_{4x+1} = M(4x,N_{4x+2})$
  - $N_{4x+2} = M(4x+2,N_{4x+3})$
  - $N_{4x+3} = M(4x+1,N_{4x+4})$
    - $N_{101} = 2$

Code

This is made by AxiomaticSystem. Like aSAN code, entries start at 0. Sometimes, it might be buggy.

aOSN Code



Sub-Notations

Oh course, there are 2 sub-notations between aSAN- and aOSN. Each level represents the oddity scale.

The Oddity Scale

- aSAN- (1): A simple array notation which reaches beyond OFP. Check out here.
- aZSN (4H): An array notation based on an idea for fixing Binary Zenith Notation. Check out here.
- aOSN-1/2 (4A/4B): A semi-oddly array notation which only takes 1 kind of exception arrays. Check out below.
- aOSN: You are here.

It doesn't mean it scales in strength. Sometimes, lower-oddly notations beat higher-oddly notations.

Semi-Oddly Strong Notations

This is between aZSN and aOSN in the Oddity scale. This level only uses Exception 1 arrays and also compares entries before N.

Red: aOSN-, Orange: aOSN-2

1. os(b,1) = b
   Base Rule
   a. os(b) = b
2. os(b,A) = os(b+1,pre(A)) (= os(b+1,A-1))
   Successor Rule
   a. Only applies when A is a successor array.
3. Standardize os(b,A) and A before beginning the process.
   Simplification
   a. Now, let A be the second entry of os(_).
4. Find the leftmost entry that is a successor array.
   Leftmost Process
   a. Let L_0 = A, p = 1, and t = 0.
   b. Let e = the p^{th} entry of L_t.
   c. If the entry (e) is 1, then move to the next entry.
      i. Increase p by 1.
      ii. Go back to Substep 4b.
   d. If the entry (e) is a Limit Array, then jump into it.
      i. Increase t by 1.
      ii. Let L_t = e **and P_t = p.**
      iii. Change p to 1.
      iv. Go back to Substep 4b.
   e. If the entry (e) is a Successor Array, then let N = L_t.
5. **Find exception arrays for all layers of N.**
   Exception Barrier
   a. **If t = 0 (or P_t doesn't exist), then skip to step 7.**
      i. **Also, let B = L_0.**
   b. **Let k = 1.**
   c. **If P_k+1-th entry of L_k > P_k+1-th entry of L_{k-1}, then L_{k-1} is an exception array.**
      (treat nonexistent entries as all 1s)
   d. **If k < t, then:**
      i. **Increase k by 1.**
      ii. **Go back to Substep 5c.**
6. Find the innermost array L_k which L_k < N.
   Layer Search / Ascender Search
   a. Let k = t.
   b. If k = 0 **or P_t doesn't exist**, then:
      a. Let B = L_0.
      b. Jump to Step 7.
   c. Decrease k by 1.
   d. If **L_k is an exception array** and L_k < N, then
      c. Let B = L_{k+1}.
      d. Jump to Step 7.
   e. **If L_k isn't an exception array,**
      entries of L_k after P_t+1th = entries of L_{k-1} after P_t+1th, and
      first P_t+1 entries of L_k < first P_t+1 entries of N,
      then:
      a. **Let B = L_{k+1}.**
      b. **Jump to Step 7.**
   f. Go back to Substep 6b.
7. Finally, we expand from B.
   Ascender Expansion
   a. Let N(n) = N, but:
      i. Change S to pre(S). (= S-1)
      ii. Change the entry of N preceding S to n.
   b. Let B(n) = B, but change N to N(n).
   c. Change B to $B^b(b)$.
   d. The process ends.