Bird's Nested Array Notation

Handles recursive functions with limit ordinal ε_0

Notes

1. This is an extension of Bird's Hyper-Dimensional Array Notation in that separator arrays (enclosed in square brackets) and dimension arrays (enclosed in angle brackets) can themselves be multidimensional or hyperdimensional arrays, or even beyond that in that the separator or dimension arrays within these are themselves multidimensional or hyperdimensional arrays, or beyond, and so on – hence these arrays are integrally embedded or 'nested' inside larger and larger versions of these arrays, and there can be any number of nested layers.
2. A, B, A_1, A_2, ... , A_k are strings of characters within separators (denoted by square brackets).
3. A_1', A_2', ... , A_k' are strings of characters within angle brackets that are identical to the strings A_1, A_2, ... , A_k respectively except that the first entries of each have been reduced by 1, e.g. if A_i (for some $1 \le i \le k$) begins with 5, A_i' begins with 4; if A_i begins with 1, A_i' begins with 0.
4. a ‹A'› b means that the space corresponding to the separator [A] is filled up with b entries of 'a' in each 1-space, b 1-spaces in each 2-space, ... , b (b-1)-spaces in each b-space (also an ω-space), b ω-spaces in each (ω+1)-space, etc. These have their own Angle Bracket Rules (see page 2).
5. # and #* are strings of characters representing the remainder of the array (if they exist).
6. The comma (,) is used as shorthand for the [1] separator.

The Nested Array Notation has 7 Main Rules and 7 Angle Bracket Rules

Main Rules

Rule M1 (only 1 or 2 entries, all in first 1-space):

{a} = a,
{a, b} = a^b.

Rule M2 (last entry in any 1-space or higher dimensional space of main array is 1):

{# [A] 1} = {#}.

When [A] < [B],

{# [A] 1 [B] #*} = {# [B] #*}.

Remove trailing 1's. See 'Algorithm for ranking two separators [A] and [B]' on pages 3-4.
Note: This Rule also applies within separator arrays.

Rule M3 (second entry is 1 or only 1 entry in first 1-space):

{a, 1 #} = a.

When [A] ≥ [2] (i.e. [A] is not a comma),

{a [A] #} = a.

Rule M4 (only 2 entries in first 1-space, next non-1 entry (n) not the first entry in its 1-space):

{a, b [A_1] 1 [A_2] ... 1 [A_k] 1, n #} = {a ‹A_1'› b [A_1] a ‹A_2'› b [A_2] ... a ‹A_k'› b [A_k] r, n-1 #},

where r = {a, b-1 [A_1] 1 [A_2] ... 1 [A_k] 1, n #}.

[A_1] ≥ [2] (i.e. non-comma). [A_1] ≥ [A_2] ≥ ... ≥ [A_k] (as a result of Rule M2).

Rule M5 (only 2 entries in first 1-space, next non-1 entry (n) is first entry in its 1-space):

{a, b [A_1] 1 [A_2] ... 1 [A_k] n #} = {a ‹A_1’› b [A_1] a ‹A_2’› b [A_2] ... a ‹A_k’› b [A_k] n-1 #}.

[A_k] ≥ [2] (i.e. non-comma). [A_1] ≥ [A_2] ≥ ... ≥ [A_k] ≥ [2] (as a result of Rule M2).

Rule M6 (Rules M1-5 do not apply, third entry is 1):

{a, b, 1, ... , 1, n #} = {a, a, a, ... , {a, b-1, 1, ... , 1, n #}, n-1 #},

where ‘1, ... , 1’ represents an unbroken string of k 1’s separated by commas (with k ≥ 1) and ‘a, ... ,’ represents an unbroken string of k-1 a’s separated by commas.

Rule M7 (Rules M1-6 do not apply):

{a, b, c #} = {a, {a, b-1, c #}, c-1 #}.

Angle Bracket Rules

Rule A1 (only 1 entry of either 0 or 1):

‘a ‹0› b’ = ‘a’,

‘a ‹1› b’ = ‘a, a, ... , a’ (with b a’s).

Rule A2 (last entry in any 1-space or higher dimensional space of array is 1):

‘a ‹# [A] 1› b’ = ‘a ‹#› b’.

When [A] < [B],

‘a ‹# [A] 1 [B] #*› b’ = ‘a ‹# [B] #*› b’.

Remove trailing 1’s. See ‘Algorithm for ranking two separators [A] and [B]’ on pages 3-4.

Rule A3 (number to right of angle brackets is 1):

‘a ‹A› 1’ = ‘a’.

Rule A4 (Rules A1-3 do not apply, first entry is 0):

‘a ‹0 [A_1] 1 [A_2] ... 1 [A_k] n #› b’ = ‘a ‹ b ‹A_1’› b [A_1] b ‹A_2’› b [A_2] ... b ‹A_k’› b [A_k] n-1 # › b’,

where n is next non-1 entry (after initial entry of 0).

Rule A5 (Rules A1-4 do not apply):

‘a ‹n #› b’ = ‘a ‹n-1 #› b [n #] a ‹n-1 #› b [n #] ... [n #] a ‹n-1 #› b’

(with b ‘a ‹n-1 #› b’ strings).

About the Nested Array Notation

Angle Bracket Rules A2, A3 and A4 look fairly similar to Main Rules M2, M3 and M5 respectively for this notation. The main differences between Rule M5 and Rule A4 (apart from the different brackets) are that in the latter rule, the array begins with 0 (rather than ‘a, b’), and the initial entries and any unbroken string of 1’s immediately after them are all replaced by ‘b ‹A_i’› b’ rather than ‘a ‹A_i’› b’ strings (that eventually become b’s rather than a’s). The first entry of an angle bracket array is the only entry that can take the value 0 – all other entries must be at least 1. The 0 is to avoid the unwieldy expressions generated when combining Rule A5 with Rule A4.

As before, all angle brackets must be removed before any of the Main (M) Rules can be executed. Rule A4 involves the creation of further angle brackets nested within the existing pair of outer angle brackets, which means that it is the newer, inner pairs of angle brackets that must be dealt with next (and each of them in turn). Only when all of these inner angle brackets are eliminated can another operation be made on the outer angle brackets.

Single-entry separators are said to have no nested layers – these are level 0 nested separators (e.g. [5]). Separator arrays that themselves contain separators (of single entry of 1 or above – the 1 indicating a comma) are level 1 nested separators (e.g. [4,4 [3] 6 [2] 3]), those that contain separator arrays that contain single-entry separators are level 2 nested separators (e.g. [3 [3 [3] 3] 3]), and so on. Angle bracket arrays similarly have nested levels, e.g. '5 ‹5 ‹5› 5› 5' is a level 2 nested angle bracket array. A main array (within curly brackets) is a level n nested array if the highest nested level of the separators and angle bracket arrays contained within the main array has level n.

Algorithm for ranking two separators [A] and [B]

The ranking of [A] and [B] is determined by the highest ranking separator within their 'base layers', then the numbers of them when they are identical. When the numbers are equal, this is repeated for the subarrays of [A] and [B] to the right of the rightmost highest ranking separator. When the highest ranking separators and their numbers within the subarrays are identical, this is repeated again for the subarrays within the subarrays, until no more separators remain (i.e. we are left with single entries). If the final entries of [A] and [B] are the same, they are deleted along with the last separators of [A] and [B], and the entire process is repeated for the truncated [A] and [B], until each of these consists of a single entry – the original [A] would be completely identical to the original [B] (symbolised as [A] = [B]) when these single entries are the same. Otherwise, either [A] ranks lower than [B] ([A] < [B]), or vice versa ([A] > [B]) – if we obtain a lower level or number for [A] than for [B] on some measure at a particular point, then the original [A] ranks lower than the original [B] and the '[A] 1' string is deleted when Rules M2 or A2 apply. (The ranking of separators within the 'base layers' of [A] and [B] are first determined by the highest ranking separator within their 'base layers' (in the second lowest layers of [A] and [B]), and so on.)

Step 1: Copy the two strings A and B as A_1 and B_1 respectively and go to Step 2.

Step 2: Copy the two strings A_1 and B_1 as A_2 and B_2 respectively and go to Step 3.

Step 3: $Lev(A_2)$ and $Lev(B_2)$ each represent the number of nested levels of the strings A_2 and B_2 respectively; a single-entry string has 0 nested levels, an 'ordinary' dimensional string (containing only single-entry separators) has 1 nested level, a string containing separators that are no higher than 'ordinary' dimensional separator arrays has 2 nested levels, and so on.

If $Lev(A_2) > Lev(B_2)$ then [A] > [B] and finish.
If $Lev(A_2) < Lev(B_2)$ then [A] < [B] and finish.
If $Lev(A_2) = Lev(B_2) > 0$ then go to Step 4.
If $Lev(A_2) = Lev(B_2) = 0$ then go to Step 7.

Step 4: Find the highest ranking separators [A*] and [B*] within the strings A_2 and B_2 respectively.

If [A*] > [B*] then [A] > [B] and finish.
If [A*] < [B*] then [A] < [B] and finish.
If [A*] = [B*] then take [H] = [A*] = [B*] and go to Step 5.

Step 5: $Num(H, A_2)$ and $Num(H, B_2)$ each represent the number of [H] separators in strings A_2 and B_2 respectively.

If $Num(H, A_2) > Num(H, B_2)$ then [A] > [B] and finish.
If $Num(H, A_2) < Num(H, B_2)$ then [A] < [B] and finish.
If $Num(H, A_2) = Num(H, B_2)$ then go to Step 6.

Step 6: Remove all entries up to and including the last [H] separator of each of the strings A_2 and B_2 and go back to Step 3.

Step 7: A_2 and B_2 each contain a single entry of a and b respectively.

If a > b then [A] > [B] and finish.
If a < b then [A] < [B] and finish.
If a = b then go to Step 8.

Step 8: Remove very last entry and separator of both A_1 and B_1.

If no entries of A_1 and B_1 remain then [A] = [B] and finish.
If any entries of A_1 and B_1 remain then go back to Step 2.

Note: It is best if we start with the top layers first (which contain the simplest separator arrays of only one entry), then work our way down, layer by layer, until we reach the bottom or base layer. (This method would help with the difficulties of ranking complex separator arrays in the lower layers.)

Hierarchy of separators

Each separator can be represented by an ordinal which denotes its level.

In Bird's Hyper-Dimensional Array Notation,

[n+1] separates n-spaces and has level n,
[1, 2] separates ω-spaces (as there is no limit on the number of dimensions) and has level ω,

and, in general,

$[n_1+1, n_2+1, n_3+1, \ldots, n_k+1]$ has level $(\omega\text{^}(k-1))n_k + \ldots + (\omega\text{^}2)n_3 + \omega n_2 + n_1$.

In Bird's Nested Array Notation,

[1 [2] 2] has level ω^ω,
[1 [3] 2] has level ω^ω^2,
[1 [4] 2] has level ω^ω^3,
[1 [n+1] 2] has level ω^ω^n,
[a+1, b+1 [n+1] c+1, d+1] (n ≥ 2) has level (ω^ω^n)(ωd+c) + ωb + a
= (ω^(ω^n+1))d + (ω^ω^n)c + ωb + a,
[a+1, b+1 [n+1] c+1 [n+1] d+1] (n ≥ 2) has level (ω^((ω^n)2))d + (ω^ω^n)c + ωb + a,
[a+1, b+1 [m+1] c+1 [n+1] d+1] (m, n ≥ 2; m < n) has level (ω^ω^n)d + (ω^ω^m)c + ωb + a,
[a+1, b+1 [m+1] c+1 [n+1] d+1] (m, n ≥ 2; m > n) has level (ω^ω^m)((ω^ω^n)d+c) + ωb + a
= (ω^(ω^m+ω^n))d + (ω^ω^m)c + ωb + a,
[1 [1, 2] 2] has level ω^ω^ω,
[a+1, b+1 [m+1, n+1] c+1, d+1] has level (ω^ω^(ωn+m))(ωd+c) + ωb + a,
[1 [1, 1, 2] 2] has level ω^ω^ω^2,
[1 [1, 1, 1, 2] 2] has level ω^ω^ω^3,
[1 [1 [2] 2] 2] has level ω^ω^ω^ω,

[1 [1 [3] 2] 2] has level ω^ω^ω^ω^2,
[1 [1 [4] 2] 2] has level ω^ω^ω^ω^3,
[1 [1 [1, 2] 2] 2] has level ω^ω^ω^ω^ω,
[1 [1 [1 [2] 2] 2] 2] has level ω^ω^ω^ω^ω^ω,
[1 [1 [1 [1, 2] 2] 2] 2] has level ω^ω^ω^ω^ω^ω^ω.

Each additional nested layer of the separator increases the height of the omega power tower level by two. [2] is a halfway house (in hyperlog to base ω terms) between a comma ([1]) and [1, 2]; this is analogous to 2 sitting halfway between 1 and ω (= 1/0) on the reciprocal scale.

Level 1 nested separators (with levels between ω and ω^ω^ω) rank higher than any single-entry (level 0 nested) separator (with level below ω), level 2 nested separators (levels between ω^^3 and ω^^5) rank higher than any level 1 nested separator, level 3 nested separators (levels between ω^^5 and ω^^7) rank higher than any level 2 nested separator. In general, level n nested separators rank higher than any level n-1 (or lower) nested separator. When the number of nested levels is the same, it is the numbers in the single-entry separators in the highest layers that determines the rank (the dimension of the level).

Since a separator can have up to ω nested levels, the limit ordinal of a separator's level in Bird's Nested Array Notation would be ω^^(2ω) = ω^^ω = ε_0, and the notation can handle recursive functions with limit ordinal ε_0. This is because ω^ε_0 = ε_0 is the limit ordinal of the number of arguments (or entries) in the array, for, in general, a separator marking an α-space can have up to ω^α arguments.

Examples

One dimensional level 1 nested arrays are the hyper-dimensional arrays (see Bird's Hyper-Dimensional Array Notation).

In the second dimension of level 1 nested arrays,

{a, b [1 [2] 2] 2} = {a ‹0 [2] 2› b}
= {a ‹b ‹1› b› b}
= {a ‹b, b, b, ... , b› b}   (with b b's inside angle brackets).

The number

{3, 2 [1 [2] 2] 2} = {3 ‹0 [2] 2› 2}
= {3 ‹2 ‹1› 2› 2}
= {3 ‹2, 2› 2}
= {3 ‹0,2› 2 [1,2] 3 ‹0,2› 2 [2,2] 3 ‹0,2› 2 [1,2] 3 ‹0,2› 2}
= {3 ‹2› 2 [1,2] 3 ‹2› 2 [2,2] 3 ‹2› 2 [1,2] 3 ‹2› 2}
= {3,3 [2] 3,3 [1,2] 3,3 [2] 3,3 [2,2] 3,3 [2] 3,3 [1,2] 3,3 [2] 3,3}.

{4, 3 [1 [2] 2] 2} = {4 ‹0 [2] 2› 3}
= {4 ‹3 ‹1› 3› 3}
= {4 ‹3, 3, 3› 3}
= {4 ‹2,3,3› 3 [3,3,3] 4 ‹2,3,3› 3 [3,3,3] 4 ‹2,3,3› 3}
= {A [2,3,3] A [2,3,3] A [3,3,3] A [2,3,3] A [2,3,3] A [3,3,3] A [2,3,3] A [2,3,3] A},

where the string of characters

A = ‘4 ‹1,3,3› 3’.

The separators used in descending order of level are [3,3,3], [2,3,3], [1,3,3], [3,2,3], [2,2,3], [1,2,3], [3,1,3], [2,1,3], [1,1,3], [3,3,2], [2,3,2], [1,3,2], [3,2,2], [2,2,2], [1,2,2], [3,1,2], [2,1,2], [1,1,2], [3,3], [2,3], [1,3], [3,2], [2,2], [1,2], [3], [2] and comma. Writing it out in full would require 3^27 = 7,625,597,484,987 4’s.

{3, 2 [2 [2] 2] 2} = {3 ‹1 [2] 2› 2}
= {3 ‹0 [2] 2› 2 [1 [2] 2] 3 ‹0 [2] 2› 2}
= {3,3 [2] 3,3 [1,2] 3,3 [2] 3,3 [2,2] 3,3 [2] 3,3 [1,2] 3,3 [2] 3,3 [1 [2] 2]
3,3 [2] 3,3 [1,2] 3,3 [2] 3,3 [2,2] 3,3 [2] 3,3 [1,2] 3,3 [2] 3,3}.

In the third dimension of level 1 nested arrays,
{a, b [1 [3] 2] 2} = {a ‹0 [3] 2› b}
= {a ‹b ‹2› b› b}
= {a ‹ b,b,...,b [2] b,b,...,b [2] ...... [2] b,b,...,b › b}
(with b ‘rows’ of b entries = b^2 entries of b inside angle brackets).

The number
{3, 2 [1 [3] 2] 2} = {3 ‹0 [3] 2› 2}
= {3 ‹2 ‹2› 2› 2}
= {3 ‹2,2 [2] 2,2› 2}
= {A [1,2 [2] 2,2] A [2,2 [2] 2,2] A [1,2 [2] 2,2] A},
where A = ‘3 ‹0,2 [2] 2,2› 2’
= ‘3 ‹2 [2] 2,2› 2’
= ‘B [1 [2] 2,2] B [2 [2] 2,2] B [1 [2] 2,2] B’,
where B = ‘3 ‹0 [2] 2,2› 2’
= ‘3 ‹2 ‹1› 2 [2] 1,2› 2’
= ‘3 ‹2,2 [2] 1,2› 2’
= ‘C [1,2 [2] 1,2] C [2,2 [2] 1,2] C [1,2 [2] 1,2] C’,
where C = ‘3 ‹0,2 [2] 1,2› 2’
= ‘3 ‹2 [2] 1,2› 2’
= ‘D [1 [2] 1,2] D [2 [2] 1,2] D [1 [2] 1,2] D’,
where D = ‘3 ‹0 [2] 1,2› 2’
= ‘3 ‹2 ‹1› 2 [2] 2 ‹0› 2› 2’
= ‘3 ‹2,2 [2] 2› 2’
= ‘E [1,2 [2] 2] E [2,2 [2] 2] E [1,2 [2] 2] E’,
where E = ‘3 ‹0,2 [2] 2› 2’
= ‘3 ‹2 [2] 2› 2’
= ‘F [1 [2] 2] F [2 [2] 2] F [1 [2] 2] F’,
where F = ‘3 ‹0 [2] 2› 2’
= ‘3 ‹2 ‹1› 2› 2’
= ‘3 ‹2, 2› 2’
= ‘3,3 [2] 3,3 [1,2] 3,3 [2] 3,3 [2,2] 3,3 [2] 3,3 [1,2] 3,3 [2] 3,3’.

The number
{4, 3 [1 [3] 2] 2} = {4 ‹0 [3] 2› 3}
= {4 ‹3 ‹2› 3› 3}
= {4 ‹ 3 ‹1› 3 [2] 3 ‹1› 3 [2] 3 ‹1› 3 › 3}
= {4 ‹ 3,3,3 [2] 3,3,3 [2] 3,3,3 › 3}

= {A [3,3,3 [2] 3,3,3 [2] 3,3,3] A [3,3,3 [2] 3,3,3 [2] 3,3,3] A},
where A = ‘4 ‹ 2,3,3 [2] 3,3,3 [2] 3,3,3 › 3’.

Examples of level 1 nested arrays are:

{3, 3 [1,2] 3},
{5, 5, 5 [2,3,4] 3 [2] 2},
{4, 4, 4, 4 [4,4] 3, 3 [3,3 [6] 3,3 [9] 3,3] 2},
{3, 3, 3 [10 [100] 10] 3, 3, 3 [10 [{3,3 [3] 3,3}] 10] 3, 3, 3}.

Level 1 nested separators include

[1,2], [3,3 [6] 3,3 [9] 3,3] and [10 [{3,3 [3] 3,3}] 10].

Level 1 nested angle bracket arrays include

‘3 ‹3, 3› 3’ and ‘3 ‹3 ‹3› 3› 3’.

Level 2 nested arrays start with

{a, b [1 [1, 2] 2] 2} = {a ‹0 [1, 2] 2› b}
= {a ‹b ‹0, 2› b› b}
= {a ‹b ‹b ‹0› b› b› b}
= {a ‹b ‹b› b› b}.

The number

{4, 3 [1 [1, 2] 2] 2} = {4 ‹0 [1, 2] 2› 3}
= {4 ‹3 ‹0, 2› 3› 3}
= {4 ‹3 ‹3› 3› 3}
= {4 ‹ 3 ‹2› 3 [3] 3 ‹2› 3 [3] 3 ‹2› 3 › 3}
= {4 ‹A› 3}

has highest separator [A] or

[3,3,3 [2] 3,3,3 [2] 3,3,3 [3] 3,3,3 [2] 3,3,3 [2] 3,3,3 [3] 3,3,3 [2] 3,3,3 [2] 3,3,3].

In the second dimension of level 2 nested arrays,

{a, b [1 [1 [2] 2] 2] 2} = {a ‹0 [1 [2] 2] 2› b}
= {a ‹b ‹0 [2] 2› b› b}
= {a ‹b ‹b ‹1› b› b› b}
= {a ‹b ‹b, b, b, ... , b› b› b}
(with b b’s inside inner angle brackets).

In the third dimension of level 2 nested arrays,

{a, b [1 [1 [3] 2] 2] 2} = {a ‹0 [1 [3] 2] 2› b}
= {a ‹b ‹0 [3] 2› b› b}
= {a ‹b ‹b ‹2› b› b› b}
= {a ‹b ‹ b,b,...,b [2] b,b,...,b [2] ...... [2] b,b,...,b › b› b}
(with b ‘rows’ of b entries = b^2 entries of b inside inner angle brackets).

The number

{4, 3 [1 [1 [3] 2] 2] 2} = {4 ‹0 [1 [3] 2] 2› 3}
= {4 ‹3 ‹0 [3] 2› 3› 3}
= {4 ‹3 ‹3 ‹2› 3› 3› 3}
= {4 ‹3 ‹ 3,3,3 [2] 3,3,3 [2] 3,3,3 › 3› 3}.

Examples of level 2 nested arrays are:

{3, 3 [3 [1,2] 3] 3},

{5, 5, 5 [4,4 [2,3,4] 3,3 [6] 2,2] 3 [2,2] 2 [9] 2},

{10, 10 [10 [10 [100] 10] 10] 10}.

In the above arrays, the level 2 nested separators are

[3 [1,2] 3], [4,4 [2,3,4] 3,3 [6] 2,2] and [10 [10 [100] 10] 10].

Level 2 nested dimension arrays include

‘3 ‹3 ‹3,3› 3› 3’ and ‘3 ‹3 ‹3 ‹3› 3› 3› 3’.

The following array

{3, 2 [5 [4 [3 [2] 3] 4] 5] 2}

is a level 3 nested array, and

[5 [4 [3 [2] 3] 4] 5]

is a level 3 nested separator. The two 5’s are said to be in the first layer of the separator, the two 4’s are said to be in the second layer, the two 3’s are said to be in the third layer and the 2 is said to be in the fourth layer (or top layer) of the separator.

When all of the numbers on the left hand side of the nested separators in the above array are 1’s and all of the numbers on the right hand side of the nested separators are 2’s, we get

{3, 2 [1 [1 [1 [2] 2] 2] 2] 2}
= {3 ‹0 [1 [1 [2] 2] 2] 2› 2}
= {3 ‹2 ‹0 [1 [2] 2] 2› 2› 2}
= {3 ‹2 ‹2 ‹0 [2] 2› 2› 2› 2}
= {3 ‹2 ‹2 ‹2 ‹1› 2› 2› 2› 2}
= {3 ‹2 ‹2 ‹2,2› 2› 2› 2}
= {3 ‹2 ‹ 2,2 [2] 2,2 [1,2] 2,2 [2] 2,2 [2,2] 2,2 [2] 2,2 [1,2] 2,2 [2] 2,2 › 2› 2}.

The dimension array within the curly brackets in the above example

‘3 ‹2 ‹2 ‹2,2› 2› 2› 2’

is a level 3 nested dimension array.

The number

{4, 3 [1 [1 [1 [3] 2] 2] 2] 2} = {4 ‹0 [1 [1 [3] 2] 2] 2› 3}
= {4 ‹3 ‹0 [1 [3] 2] 2› 3› 3}
= {4 ‹3 ‹3 ‹0 [3] 2› 3› 3› 3}
= {4 ‹3 ‹3 ‹3 ‹2› 3› 3› 3› 3}
= {4 ‹3 ‹3 ‹ 3,3,3 [2] 3,3,3 [2] 3,3,3 › 3› 3› 3}.

As the level of nested arrays get higher and higher, some of the separator arrays become ‘trees’ of arrays, which branch off into further separator arrays, which, in turn, have smaller branches of more separator arrays, until we get ‘leaves’ (separators with single entries). This is an incredibly powerful notation that enables us to generate ever huger and huger numbers – something that is unlikely to be surpassed by any other notation.

The Nested Array Notation notation allows us to create numbers as huge as:

{3, 3 [3, 3 [3, 3 [3, 3 [3, 3 [3, 3 [3, 3 [3, 3 [3,3 [3] 2] 2] 2] 2] 2] 2] 2] 2] 2}

(a level 8 nested array)

and

{3 ‹3 ‹3 ‹3 ‹3 ‹3 ‹3 ‹3 ‹3 ‹3› 3› 3› 3› 3› 3› 3› 3› 3› 3}

(a level 8 nested dimension array – 10 3's from centre out),

and much, much larger, of course – as the separators can grow into 'trees', quite literally! And what is more, is that evaluating those 'tree' arrays brings an almost infinite 'forest' of them before you know it! Perhaps, these arrays ought be termed 'forest' arrays!

The function,

f(n) = {n, n [1 [1 [... [1 [2] 2] ...] 2] 2] 2} (with (n+1)/2 square brackets, n odd)

= {n, n [1 [1 [... [1 [1, 2] 2] ...] 2] 2] 2} (with n/2 square brackets, n even)

is ε_0-recursive, as ε_0 = ω^^ω.

The Fusible Numbers (http://www.mathpuzzle.com/fusible.pdf) are ε_0-recursive as they grow as rapidly as f(n) in the paragraph above. In that paper, m(x) is the difference between x and smallest fusible number > x.

If x < 0, then m(x) = -x, otherwise, m(x) = m(x - m(x-1))/2.

The sequence of numbers f(x) = $-\log_2$ m(x) grows as follows:

f(0) = 1,
f(1) = 3,
f(2) = 10,
f(3) = 1,541,023,937,
f(4) is probably between Skewes' Number and Graham's Number.

The Kirby-Paris hydra game (http://googology.wikia.com/wiki/Kirby-Paris_hydra) is a one-player game played on a tree. The game is played as follows:

1. We start with a rooted tree T. Call its root r.
2. At step n (starting with 1), Hercules picks a leaf vertex of the tree. Call the leaf a and its parent b:
   A. a is deleted from T.
   B. If b = r, nothing happens. Otherwise, let c be the parent of b. Consider the subtree consisting of b and all its children; copy this subtree n times. Attach all these copies to c.

The hydra grows very rapidly at first, but regardless of what strategy Hercules uses, the hydra will eventually reduce to just a single node. Define Hydra(n) as the number of turns it takes Hercules to defeat a hydra consisting of a path of length n, assuming he always cuts the rightmost edge each step. Then

Hydra(0) = 0,
Hydra(1) = 1,
Hydra(2) = 3,
Hydra(3) = 37,
Hydra(4) > {5, 5, 4, 3},
Hydra(5) > {5, 5, 5, 5, 5 [2] 5, 5, 5, 5, 5 [2] 5, 5, 5, 5},
Hydra(6) > {5, 5 [5, 3] 2},
Hydra(7) > {5, 5 [1 [2] 1 [2] 1,1,1,1,2] 2},
Hydra(8) > {5, 5 [1 [5, 3] 2] 2},
Hydra(9) > {5, 5 [1 [1 [2] 1 [2] 1,1,1,1,2] 2] 2}.

In general, for n ≥ 6,

Hydra(n) > {5, 5 [1 [1 [... [1 [2] 1 [2] 1,1,1,1,2] ...] 2] 2] 2}

(with (n-3)/2 layers of square brackets, n odd)

> {5, 5 [1 [1 [... [1 [5, 3] 2] ...] 2] 2] 2}

(with (n-4)/2 layers of square brackets, n even), so, the growth rate of Hydra(n) is also at the ε_0 level of recursion.

Another example of an ε_0-recursive function is the Goodstein sequence (G(n)). Some values are shown below:

G(0) = 1,
G(1) = 2,
G(2) = 4,
G(3) = 6,
G(4) = $3\times2^{402,653,211}$ - 2,
10^^10^^10^^(10^10^10^20) < G(5) < 4^^^8 = {4, 8, 3},
{2, 3, {2, 3, $3\times2^{402,653,211}$}} < G(8) < {3, 4, 1, 2},
{3, 4, 3, 3, 3} < G(16) < {3, 5, 3, 3, 3},
{3, 4, 2 [2] 2} < G(32) < {3, 5, 2 [2] 2},
{3, 4, 4 [2] 2} < G(64) < {3, 5, 4 [2] 2},
{3, 4, 1, 2 [2] 2} < G(128) < {3, 5, 1, 2 [2] 2},
{3, 5 [2] 3} < G(256) < {3, 6 [2] 3},
{3, 3 [4] 2} < G(65,536) < {3, 4 [4] 2},
{3, 3 [4, 4, 4, 3] 3} < G(2^65,536) < {3, 4 [1 [2] 2] 2},
{3, 3 [1 [1, 2] 2] 2} < G(2^2^65,536) < {3, 3 [1 [1 [2] 2] 2] 2},
{3, 3 [1 [1 [2] 2] 2] 2} < G(2^2^2^65,536) < {3, 3 [1 [1 [1, 2] 2] 2] 2}.

Further examples can be seen at http://googology.wikia.com/wiki/Goodstein_sequences.

Author: Chris Bird (Gloucestershire, England, UK)
Last modified: 31 August 2013
E-mail: m.bird44 at btinternet.com (not clickable to thwart spambots!)