Bird's Hyper-Dimensional Array Notation

Handles recursive functions with limit ordinal $\omega^{\omega^{\omega}}$

Notes

1. This is an extension of Bird's Multi-Dimensional Array Notation in that it goes into 'hyperdimensions'.
2. [a,b,c,...] is a separator array (enclosed in square brackets) marking the end of an (a-1,b,c,...)-space of the main array (enclosed in curly brackets). For example, [2] marks the end of a 1-space or 'row', [3] marks the end of a a 2-space or 'plane', and so on; [1,2] marks the end of an 0-space in the first 'hyperdimension', [2,2] marks the end of a 1-space in the first 'hyperdimension', and so on; [1,3] marks the end of an 0-space in the second 'hyperdimension'; [1,1,2] marks the end of an 0-space in the first 'hyper-2' dimension, [1,1,1,2] marks the end of an 0-space in the first 'hyper-3' dimension, and so on. An array where the 'highest ranking' separator is [a,b,c,...] is an (a,b,c,...)-dimensional array.
3. 'a ‹c,d,e,...› b' angle bracket array strings (where the 'c,d,e,...' within the angle brackets represents the dimension) have their own set of rules (see Angle Bracket Rules on page 2).
4. # and #* are strings of characters representing the remainder of the array (if they exist).
5. The comma (,) is used as shorthand for the [1] separator.

The Hyper-Dimensional Array Notation has 7 Main Rules and 5 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,b,c,...] 1} = {#}.

When $[a_1,b_1,c_1,..] < [a_2,b_2,c_2,..]$,

{# $[a_1,b_1,c_1,..]$ 1 $[a_2,b_2,c_2,..]$ #*} = {# $[a_2,b_2,c_2,..]$ #*}.

In separator strings,

[a,b,c,...,z,1] = [a,b,c,...,z].

Remove trailing 1's. See 'Rankings of separators' on pages 3-4.

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

{a, 1 #} = a.

When $[a_1,b_1,c_1,..] \geq [2]$ (in other words, $[a_1,b_1,c_1,..]$ is not a comma),

{a $[a_1,b_1,c_1,..]$ #} = 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 $[a_1,b_1,c_1,..]$ 1 $[a_2,b_2,c_2,..]$ .... 1 $[a_k,b_k,c_k,..]$ 1, n #}

= {a ‹a_1-1,b_1,c_1,..› b $[a_1,b_1,c_1,..]$ a ‹a_2-1,b_2,c_2,..› b $[a_2,b_2,c_2,..]$ ....

a ‹a_k-1,b_k,c_k,..› b $[a_k,b_k,c_k,..]$ R, n-1 #},

where R = {a, b-1 $[a_1,b_1,c_1,..]$ 1 $[a_2,b_2,c_2,..]$ .... 1 $[a_k,b_k,c_k,..]$ 1, n #}.

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,b_1,c_1,..] 1 [a_2,b_2,c_2,..] .... 1 [a_k,b_k,c_k,..] n #}

= {a ‹a_1-1,b_1,c_1,..› b [a_1,b_1,c_1,..] a ‹a_2-1,b_2,c_2,..› b [a_2,b_2,c_2,..] ....
a ‹a_k-1,b_k,c_k,..› b [a_k,b_k,c_k,..] n-1 #}.

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

In Main Rules M4 and M5, one or more ‘dimension’ array strings (within angle brackets) of the form ‘a ‹c,d,e,...› b’ are created within the main array, and each of these must be replaced by a longer string, which usually contains further ‘angle bracket’ strings, which in turn must be replaced by longer strings, and so on, until all the angle brackets have been eliminated, for example, by replacing each ‘a ‹1› b’ by ‘a, a, ... , a’ (with b a’s). The procedure for doing this has its own set of rules – the Angle Bracket Rules – where the ‘=’ sign means ‘is replaced by’. Only when all the angle brackets have been eliminated is it possible to go back to the Main Rules. The Angle Bracket Rules are shown below.

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 is 1):

‘a ‹c, d, e, ... , z, 1› b’ = ‘a ‹c, d, e, ... , z› b’ (remove trailing 1’s).

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

‘a ‹c, d, e, ...› 1’ = ‘a’.

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

‘a ‹0, 1, ... , 1, c, d, ...› b’ = ‘a ‹b, b, ... , b, c-1, d, ...› b’,

where ‘1, ... , 1’ represents an unbroken string of zero or more 1’s separated by commas and ‘b, ... , b’ represents an unbroken string of the same number of b’s separated by commas.

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

‘a ‹c, d, e, ...› b’ = ‘a ‹c-1,d,e,..› b [c,d,e,..] a ‹c-1,d,e,..› b [c,d,e,..] .... [c,d,e,..] a ‹c-1,d,e,..› b’
(with b ‘a ‹c-1,d,e,..› b’ strings).

As with the Main Rules, it is helpful when the rules are considered in sequence; first use Rule A1 if it applies, if not then use Rule A2, etc.

The Angle Bracket Rules look fairly similar to the five rules of Bird’s Linear Array Notation, except that in Rule A4, the initial 0 and any unbroken string of 1’s after the 0 are all replaced by b’s (rather than a's), with the final 1 not being replaced by an almost identical copy of the entire array. 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.

Rankings of separators

Len(A) and Len(B) are defined to be the number of entries in the separator arrays [A] and [B] respectively when all entries are separated by commas.

If Len(A) > Len(B) then [A] > [B].
If Len(A) < Len(B) then [A] < [B].
If Len(A) = Len(B) then take
n = Len(A) = Len(B),
[A] = [a_1, a_2, a_3, ... , a_n],
[B] = [b_1, b_2, b_3, ... , b_n],
where a_n, b_n ≥ 2:
If $a_n > b_n$ then [A] > [B],
if $a_n < b_n$ then [A] < [B],
if $a_n = b_n$, $a_{n-1} > b_{n-1}$ then [A] > [B],
if $a_n = b_n$, $a_{n-1} < b_{n-1}$ then [A] < [B],
if $a_n = b_n$, $a_{n-1} = b_{n-1}$, $a_{n-2} > b_{n-2}$ then [A] > [B],
if $a_n = b_n$, $a_{n-1} = b_{n-1}$, $a_{n-2} < b_{n-2}$ then [A] < [B],
and so on, until
if $a_n = b_n$, $a_{n-1} = b_{n-1}$, $a_{n-2} = b_{n-2}$, ... , $a_2 = b_2$, $a_1 > b_1$ then [A] > [B],
if $a_n = b_n$, $a_{n-1} = b_{n-1}$, $a_{n-2} = b_{n-2}$, ... , $a_2 = b_2$, $a_1 < b_1$ then [A] < [B].
If, however,
$a_n = b_n$, $a_{n-1} = b_{n-1}$, $a_{n-2} = b_{n-2}$, ... , $a_2 = b_2$, $a_1 = b_1$ then [A] = [B].

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

The comma (shorthand for [1]) separates simple entries (0-dimensional) and has level 0,
[2] separates 1-spaces and has level 1,
[3] separates 2-spaces and has level 2,
[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,
[2, 2] separates (ω+1)-spaces and has level ω+1,
[n+1, 2] separates (ω+n)-spaces and has level ω+n,
[1, 3] separates (ω2)-spaces and has level ω2,
[m+1, n+1] separates (ωn+m)-spaces and has level ωn+m,
[1, 1, 2] separates (ω^2)-spaces and has level ω^2,
[1, 1, 1, 2] separates (ω^3)-spaces and has level ω^3.

Note that ω+n > ω and ωn > ω but n+ω = ω and nω = ω for finite n.

In general,

[n_1+1, n_2+1, n_3+1, ... , n_k+1] has level (ω^(k-1))n_k + ... + (ω^2)n_3 + ωn_2 + n_1.

If the separator arrays [A] and [B] (as defined above) have levels of α and β respectively (where α and β are ordinals expressed in terms of ω and finite numbers) then

[A] > [B] when α > β,

[A] < [B] when α < β and
[A] = [B] when α = β.

The [1, 2] separator ranks higher than any separator containing a single entry (e.g. [n]) since it marks an ω-space (ω denotes an infinity) and can 'spawn' any [n] for finite n, whereas [n] merely marks an (n-1)-space. Similarly, the [1, 1, 2] separator beats any [m, n] separator since ω^2 > ω(n-1) + m-1 for all finite m and n.

Bird's Hyper-Dimensional Array Notation handles recursive functions with limit ordinal ω^ω^ω since this is the limit ordinal of the number of arguments (or entries) in the array. The [2] separator marks a 1-space, which can have up to ω arguments; the [3] separator marks a 2-space, which can have up to ω^2 arguments; the [1, 2] separator marks an ω-space, which can have up to ω^ω arguments. In general, a separator marking an α-space can have up to ω^α arguments, so the limit ordinal of the number of arguments in Hyper-Dimensional arrays is the limit ordinal of the number of arguments in the space marked by the [1, 1, ... , 1, 2] separator (with ω 1's), which marks a (ω^ω)-space.

Examples

In the first 'hyperdimension',
{a, b [1, 2] 2} = {a ‹0, 2› b}
= {a ‹b› b} (b dimensional b^b array of a's).

For example,
{3, 2 [1, 2] 2} = {3 ‹0, 2› 2}
= {3 ‹2› 2}
= {3, 3 [2] 3, 3},
{3, 2, 2 [1, 2] 2} = {3, 3 [1, 2] 2}
= {3 ‹3› 3}
= {3 ‹2› 3 [3] 3 ‹2› 3 [3] 3 ‹2› 3}
= {3‹1›3 [2] 3‹1›3 [2] 3‹1›3 [3] 3‹1›3 [2] 3‹1›3 [2] 3‹1›3 [3] 3‹1›3 [2] 3‹1›3 [2] 3‹1›3}
= {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}
= N (gigantic number when the 3-dimensional 3^3 array of 3's is evaluated),
{3, 3, 2 [1, 2] 2} = {3, {3, 2, 2 [1, 2] 2} [1, 2] 2}
= {3, N [1, 2] 2}
= {3 ‹0, 2› N}
= {3 ‹N› N}
is an N-dimensional N^N array of 3's. Not only are there N dimensions but there are N 3's in each line, N lines in each plane, and so on, up to N (N-1)-spaces in the array!

The number,
{4, 3 [1, 2] 2} = {4 ‹0, 2› 3}
= {4 ‹3› 3}
= {4 ‹2› 3 [3] 4 ‹2› 3 [3] 4 ‹2› 3}
= {4,4,4 [2] 4,4,4 [2] 4,4,4 [3] 4,4,4 [2] 4,4,4 [2] 4,4,4 [3] 4,4,4 [2] 4,4,4 [2] 4,4,4}
= {A},
where A represents the string of characters in the array (in order to avoid unwieldy arrays in the further examples below).

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

In the second ‘hyperdimension’,
{4, 3 [1, 3] 2} = {4 ‹0, 3› 3}
= {4 ‹3, 2› 3}
= {4 ‹2,2› 3 [3,2] 4 ‹2,2› 3 [3,2] 4 ‹2,2› 3}
= {B [3,2] B [3,2] B}
= {C},
{4, 3 [2, 3] 2} = {4 ‹1, 3› 3}
= {4 ‹0,3› 3 [1,3] 4 ‹0,3› 3 [1,3] 4 ‹0,3› 3}
= {C [1,3] C [1,3] C},
{4, 3 [3, 3] 2} = {4 ‹2, 3› 3}
= {4 ‹1,3› 3 [2,3] 4 ‹1,3› 3 [2,3] 4 ‹1,3› 3}
= {C [1,3] C [1,3] C [2,3] C [1,3] C [1,3] C [2,3] C [1,3] C [1,3] C}
= {D}.

In the first ‘hyper-2’ dimension,
{4, 3 [1, 1, 2] 2} = {4 ‹0, 1, 2› 3}
= {4 ‹3, 3› 3}
= {4 ‹2,3› 3 [3,3] 4 ‹2,3› 3 [3,3] 4 ‹2,3› 3}
= {D [3,3] D [3,3] D}.

In the first ‘hyper-3’ dimension,
{4, 3 [1, 1, 1, 2] 2} = {4 ‹0, 1, 1, 2› 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}
= {S [2,3,3] S [2,3,3] S [3,3,3] S [2,3,3] S [2,3,3] S [3,3,3] S [2,3,3] S [2,3,3] S},
where the string of characters
S = ‘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.

Author: Chris Bird (Gloucestershire, England, UK)
Last modified: 2 August 2017
E-mail: m.bird44 at btinternet.com (not clickable to thwart spambots!)