Bird's Linear Array Notation

Handles recursive functions with limit ordinal ω

The Linear Array Notation has 5 rules of operation

Rule 1 (only 1 or 2 entries):

{a} = a,

{a, b} = a^b.

Rule 2 (last entry is 1):

{a, b, c, ... , z, 1} = {a, b, c, ... , z}   (remove trailing 1's).

Rule 3 (second entry is 1):

{a, 1, c, d, ... , z} = a.

Rule 4 (third entry is 1):

{a, b, 1, ... , 1, d, e, ... , z} = {a, a, a, ... , {a, b-1, 1, ... , 1, d, e, ... , z}, d-1, e, ... , z}.

The '...' between the 1's represents an unbroken string of 1's – there can be any number of 1's, from one 1 (third entry alone) to a string of 1's up to the penultimate entry – it is the last 1 of this unbroken string (not necessarily the last 1 in the array) that is replaced by a copy of the entire array with its second entry reduced by 1, and all entries prior to this become an unbroken string of a's. This is the only way that a fourth or subsequent entry in the array can be reduced in number (albeit by 1); if there are n 1's in the unbroken string from the third entry onwards then the (n+3)th entry (represented by d) is reduced by 1.

Rule 5 (rules 1-4 do not apply):

{a, b, c, d, ... , z} = {a, {a, b-1, c, d, ... , z}, c-1, d, ... , z}.

The second entry is replaced by a copy of the entire array with its second entry reduced by 1, in order to reduce the third entry by 1.

It is helpful when the rules are considered in sequence; first use Rule 1 if it applies, if not then use Rule 2, etc. If none of Rules 1-4 apply then Rule 5 will. The curly brackets can only be removed after the array inside the curly brackets has been evaluated into a single number.

About the Linear Array Notation

Bird's Linear Array Notation is similar to Jonathan Bowers' Array Notation for linear arrays except that he originally defined {a, b} = a+b rather than {a, b} = a^b. It is more logical to set {a, b} = a^b rather than {a, b} = a+b because Rules 2 and 3 would then work for arrays with 2 entries, just as they do for arrays with 3 or more entries. For example, {a, 1} = a^1 = a, whereas a+1 ≠ a.

Jonathan Bowers' Array Notation builds on his Extended Operator Notation, which was originally

a {1} b = a + b,

a {2} b = a × b,

a {3} b = a ^ b,

and, in general,

a {c} b = a {c-1} (a {c-1} (a {c-1} (... (a {c-1} a)...))) (with b terms)
= a {c-1} (a {c} (b-1)).

Bird’s Linear Array Notation modifies this so that
a {1} b = a ^ b,
a {2} b = a ^ (a ^ (a ^ (a ^ (... (a ^ a)...)))) (with b terms),
a {3} b = a {2} (a {2} (a {2} (... (a {2} a)...))) (with b terms),
and so on.

Bowers nested these huge numbers inside operators by defining
a {{1}} 2 = a {a} a,
a {{1}} 3 = a {a {a} a} a,
a {{1}} 4 = a {a {a {a} a} a} a,
a {{1}} b = a {a {a { ... {a {a} a} ... } a} a} a (with b a’s from centre out),
and
a {{c}} b = a {{c-1}} (a {{c-1}} (a {{c-1}} (... (a {{c-1}} a)...))) (with b terms).

Further, he defined
a {{{1}}} b = a {{a {{a {{ ... {{a {{a}} a}} ... }} a}} a}} a
(with b a’s from centre out),
a {{{c}}} b = a {{{c-1}}} (a {{{c-1}}} (a {{{c-1}}} (... (a {{{c-1}}} a)...)))
(with b terms),
a {{{{1}}}} b = a {{{a {{{a {{{ ... {{{a {{{a}}} a}}} ... }}} a}}} a}}} a
(with b a’s from centre out),
a {{{{c}}}} b = a {{{{c-1}}}} (a {{{{c-1}}}} (a {{{{c-1}}}} (... (a {{{{c-1}}}} a)...)))
(with b terms),
and, in general, when $\{\}_d$ denotes {{{..{}..}}} with d pairs of curly brackets,
$a \{1\}_d\ b = a \{a \{a \{ \ldots \{a \{a\}_{d-1}\ a\}_{d-1} \ldots \}_{d-1}\ a\}_{d-1}\ a\}_{d-1}\ a$ (with b a’s from centre out),
$a \{c\}_d\ b = a \{c-1\}_d\ (a \{c-1\}_d\ (a \{c-1\}_d\ (\ldots (a \{c-1\}_d\ a)\ldots)))$ (with b terms).

The last 2 equations can be rewritten as follows:
$a \{1\}_d\ b = a \{a \{1\}_d\ (b-1)\}_{d-1}\ a$,
$a \{c\}_d\ b = a \{c-1\}_d\ (a \{c\}_d\ (b-1))$.

In an array of 3 entries,
{a, b, c} = a {c} b
= a ^^^···^ b (with c Knuth’s up-arrows)
= a → b → c (in Conway’s Chained Arrow Notation).

In an array of 4 entries,
{a, b, c, d} = a {{{..{c}..}}} b (with d pairs of curly brackets).

This is because, in the case of arrays with 3 entries,
{a, b, 1} = a {1} b
= a ^ b
= {a, b} (gives Rule 2 for 3-entry arrays),
{a, 1, c} = a {c} 1
= a (gives Rule 3 for 3-entry arrays),

$\{a, b, c\} = a \{c\} b$
$= a \{c\text{-}1\} (a \{c\} (b\text{-}1))$
$= \{a, (a \{c\} (b\text{-}1)), c\text{-}1\}$
$= \{a, \{a, b\text{-}1, c\}, c\text{-}1\}$ (gives Rule 5 for 3-entry arrays).

In the case of arrays with 4 entries (with $\{\}_d$ denoting $\{\{\{..\{\}..\}\}\}$ with d pairs of curly brackets),
$\{a, b, c, 1\} = a \{c\} b$
$= \{a, b, c\}$ (gives Rule 2 for 4-entry arrays),
$\{a, 1, c, d\} = a \{c\}_d 1$
$= a$ (gives Rule 3 for 4-entry arrays),
$\{a, b, 1, d\} = a \{1\}_d b$
$= a \{a \{1\}_d (b\text{-}1)\}_{d\text{-}1} a$
$= \{a, a, (a \{1\}_d (b\text{-}1)), d\text{-}1\}$
$= \{a, a, \{a, b\text{-}1, 1, d\}, d\text{-}1\}$ (gives Rule 4 for 4-entry arrays),
$\{a, b, c, d\} = a \{c\}_d b$
$= a \{c\text{-}1\}_d (a \{c\}_d (b\text{-}1))$
$= \{a, (a \{c\}_d (b\text{-}1)), c\text{-}1, d\}$
$= \{a, \{a, b\text{-}1, c, d\}, c\text{-}1, d\}$ (gives Rule 5 for 4-entry arrays).

The Linear Array Notation with n entries ($n \geq 2$) handles fast-growing functions up to recursion level n-1 (or (n-1)-recursive functions) since there are n-1 arguments (excluding the first entry, which is the base or 'filler' entry). Knuth's Up-arrow Notation only goes up to recursion level 2 because it does not extend beyond 3-entry arrays. Conway's Chained Arrow Notation only goes up to recursion level 3 as it does not extend beyond 4-entry arrays. Since there is no limit to the number of entries allowed in Bird's Linear Array Notation, it extends upwards to recursion level ω (the smallest infinite ordinal). In other words, the Linear Array Notation handles recursive functions with limit ordinal ω, which translates to limit ordinals of ω^ω and ω^ω^ω in the fast-growing and Hardy hierarchies of functions respectively.

Examples

Using Bird's Linear Array Notation and Bird's Extended Operator Notation,
$\{3, 3, 1\} = \{3, 3\}$
$= 3 \wedge 3$
$= 27$,
$\{3, 3, 2\} = 3 \{2\} 3$
$= 3 \wedge (3 \wedge 3)$ (since $a \{1\} b = a \wedge b$)
$= 3 \wedge 27$
$= 7{,}625{,}597{,}484{,}987$,
$\{3, 3, 3\} = 3 \{3\} 3$
$= 3 \{2\} (3 \{2\} 3)$
$= 3 \{2\} (3 \wedge (3 \wedge 3))$
$= 3 \{2\} 7{,}625{,}597{,}484{,}987$
$= 3 \wedge (3 \wedge (3 \wedge (\ldots (3 \wedge 3)\ldots)))$
(a power tower with 7,625,597,484,987 terms – even if every 3 in the stack was as small as the thickness of a human hair, the tower would reach the moon and back),

{3, 3, 4} = 3 {4} 3
= 3 {3} (3 {3} 3)
= 3 {3} (3 ^ (3 ^ (3 ^ (... (3 ^ 3)...)))) (with 7,625,597,484,987 3's in power tower)
= 3 {2} (3 {2} (3 {2} (... (3 {2} 3)...)))
(where the number of terms is a power tower of 7,625,597,484,987 3's).

While the number
{3, 2, 1, 2} = 3 {{1}} 2
= 3 {3} 3,
the number
{3, 3, 1, 2} = 3 {{1}} 3
= 3 {3 {3} 3} 3
and the number
{3, 4, 1, 2} = 3 {{1}} 4
= 3 {3 {3 {3} 3} 3} 3.

Since
{3, 65, 1, 2} = 3 {{1}} 65
= 3 {3 {3 { ... {3 {3} 3} ... } 3} 3} 3 (with 65 3's from centre out)
and Graham's Number is achieved by changing the 3 in the centre to a 4, it follows that
{3, 65, 1, 2} < (Graham's Number) << {3, 66, 1, 2}.

While the number
{3, 2, 2, 2} = 3 {{2}} 2
= 3 {{1}} 3
= 3 {3 {3} 3} 3
= {3, 3, 1, 2},
the number
{3, 3, 2, 2} = 3 {{2}} 3
= 3 {{1}} (3 {{1}} 3)
= 3 {{1}} (3 {3 {3} 3} 3)
= {3, (3 {3 {3} 3} 3), 1, 2}
= {3, {3, 3, 1, 2}, 1, 2},
and so is very much larger than Graham's Number.

While the number
{3, 2, 3, 2} = 3 {{3}} 2
= 3 {{2}} 3
= {3, 3, 2, 2},
the number
{3, 3, 3, 2} = 3 {{3}} 3
= 3 {{2}} (3 {{2}} 3)
= {3, (3 {{2}} 3), 2, 2}
= {3, {3, 3, 2, 2}, 2, 2}.

While the number
{3, 2, 1, 3} = 3 {{{1}}} 2
= 3 {{3}} 3
= {3, 3, 3, 2},

the number

{3, 3, 1, 3} = 3 {{{1}}} 3
= 3 {{3 {{3}} 3}} 3
= {3, 3, (3 {{3}} 3), 2}
= {3, 3, {3, 3, 3, 2}, 2}.

Since

{a, b, c, d} = a {{{..{c}..}}} b (with d pairs of curly brackets),

the following numbers can be written as follows:

{3, 3, 3, 3} = 3 {{{3}}} 3,
{10, 10, 10, 10} = 10 {{{{{{{{{{10}}}}}}}}}} 10,
{10, 10, 100, 20} = 10 {{{{{{{{{{{{{{{{{{{{100}}}}}}}}}}}}}}}}}}}} 10.

When the number of curly brackets in the Extended Operator Notation becomes large, it is easier to use the Linear Array Notation.

While the number represented by

{a, 2, 1, 1, 2} = {a, a, a, a}
= a {{{..{a}..}}} a (with a pairs of curly brackets),

that represented by

{a, 3, 1, 1, 2} = {a, a, a, {a, 2, 1, 1, 2}}
= a {{{..{a}..}}} a (with a {{{..{a}..}}} a pairs of curly brackets (with a pairs of curly brackets))

and

{a, 4, 1, 1, 2} = {a, a, a, {a, 3, 1, 1, 2}}
= a {{{..{a}..}}} a (with a {{{..{a}..}}} a pairs of curly brackets (with a {{{..{a}..}}} a pairs of curly brackets (with a pairs of curly brackets))).

In general,

{a, b, 1, 1, 2} = {a, a, a, {a, b-1, 1, 1, 2}}
= a {{{..{a}..}}} a (with {a, b-1, 1, 1, 2} pairs of curly brackets).

Hence,

{3, 5, 1, 1, 2} = 3 {{{..{3}..}}} 3 (with 3 {{{..{3}..}}} 3 pairs of curly brackets (with 3 {{{..{3}..}}} 3 pairs of curly brackets (with 3 {{{3}}} 3 pairs of curly brackets))).

Friedman's n(k) function for k-character sequences in his Block Subsequence Theorem grows so rapidly that it approaches the limits of Bird's Linear Array Notation. While

n(1) = 3,
n(2) = 11,
n(3) > {2, 158386, 7197},
n(4) > {3, {2, 187196, 187195}, 1, 2},

the growth rate of n(k) is broadly comparable to the function

f(n) = {3, 3, 3, ... , 3} (with n entries).

Jonathan Bowers' Array Notation can be visited at: http://www.polytope.net/hedrondude/array.htm

Author: Chris Bird (Gloucestershire, England, UK)
Last modified: 1 April 2012
E-mail: m.bird44 at btinternet.com (not clickable to thwart spambots!)