

巨大数論

ふいつしゅつしゅ fish@gyafun.jp

PDF出版: <http://gyafun.jp/ln/>

最終更新日: 2009年3月2日

目次

第 1 章	はじめに	3
第 2 章	グラハム数を超えて	4
2.1	グラハム数	4
2.2	ふいつしゅ数	6
2.3	ふいつしゅ数誕生の経緯	7
第 3 章	2 重帰納関数	12
3.1	巨大数と巨大関数	12
3.2	アッカーマン関数と原始帰納関数	12
3.2.1	アッカーマン関数	12
3.2.2	原始帰納関数	13
3.2.3	2 重帰納関数	14
3.2.4	グラハム数とふいつしゅ数	15
3.3	チェーン表記	16
3.3.1	定義と計算	16
3.3.2	2 重帰納	17
3.3.3	F_1 とチェーン表記の比較	18
第 4 章	3 重帰納関数	19
4.1	3 変数アッカーマン関数	19
4.2	チェーンの長さを変数化	20
4.3	ふいつしゅ数バージョン 2	20
第 5 章	多重帰納関数	23
5.1	n 変数 Ackermann 関数	23
5.2	多重帰納に見えてそうでない関数	23
5.3	矢印回転表記とバード数	24
5.3.1	矢印回転表記	24
5.3.2	バード数	26
5.3.3	バード数の大きさ	27
5.3.4	バード数とアッカーマンの比較	27
5.4	ふいつしゅ数バージョン 3	28
5.5	巨大数の比較	30

第 6 章	順序数と Hardy 関数	32
6.1	多重帰納を超えて	32
6.2	順序数	32
6.3	Hardy 関数	33
6.4	色々な関数の Hardy による近似	35
第 7 章	証明可能帰納関数	36
7.1	証明可能帰納関数	36
7.2	2 重リストアッカーマン関数	36
7.3	ふいつしゅ数バージョン 3 の拡張	37
7.4	多重リストアッカーマン関数	38
7.5	ふいつしゅ数バージョン 5	38
7.6	ヒドラゲーム	41
第 8 章	帰納関数	42
8.1	巨大関数論から巨大順序数論へ	42
8.2	ω_0 以下の順序数	42
8.3	ふいつしゅ数バージョン 6	43
8.4	Veblen 関数	46
8.5	拡張 Veblen 関数	47
8.6	Taranovsky の順序数記法	48
8.6.1	多変数 C_0	48
8.6.2	2 重リスト C_0 以上	49
8.7	その他の関数	49
8.7.1	ナゴヤ関数	49
8.7.2	アルカ数	49
8.8	帰納関数の大きさ比較	49
第 9 章	計算不可能な関数	51
9.1	ビジービーバー関数	51
9.1.1	チューリングマシン	51
9.2	ふいつしゅ数バージョン 4	51
9.3	ビジービーバーの Hardy 的拡張	51
9.4	帰納的でない順序数と Hardy 関数	52

第1章 はじめに

私たちが日常生活で使う大きな数は、たとえば世界の人口が66億人いるとか、国家予算が200兆円であるといったように、100兆=10の14乗程度までです。その上の単位である京を知っていても、さらにその上の垓といった単位を使うことはめったにありません。

科学の世界では、たとえば1モルに含まれる要素粒子の数を表すアボガドロ数という数は、約 6.023×10^{23} 個で、これは約6023垓個と書くこともできます。物理的に意味のある非常に大きな定数としては、エディントン数という数があります。これは、Eddingtonが、全宇宙に存在する陽子の正確な数であると定義したもので、 $136 \times 2^{256} = 1.575 \times 10^{79}$ です。有名な「無量大数」という数の単位は 10^{68} で、それよりもエディントン数の方が大きいことになります。「天文学的数字」という表現がありますが、天文学の世界でも、エディントン数よりも大きい数を扱うことはまずありません。

ところが、それよりもさらに大きな数が、仏教の経典には書かれているとのこと。大方広仏華嚴經の巻第四十五、阿僧祇品第三十という経典に書かれている不可説不可説転という数の大きさを計算すると、 $10^7 \times 2^{122}$ となり、これは $10^{37218383881977644441306597687849648128}$ というとてつもない大きな数です。「不可説不可説転メートル」がどれくらい長い距離なのか、あるいは「不可説不可説転秒」がどれくらい長い時間なのか、想像をすることすら不可能です。

一方、数学の世界では、不可説不可説転とは比べ物にならないほど大きなグラハム数という数が定義されています。これは、ギネスブックにも載ったそうです。

1970年に、グラハム・ロートシルトは「 n 次元の超立方体の合計 2^n 個の頂点を総て結び、それを赤と青の二色の何れかに塗る。このとき n が充分大きいならば、どのような塗り方をしても、必ず同一平面上にある四点でそれらを結ぶ線が総て同一の色であるものが存在する」という定理を発表しました。この定理の中の「十分大きい n 」の1つがグラハム数であるとのこと。

この数の大きさは、たとえば「10の10乗の10乗の...と100回繰り返して...」といったような表現では、とても表現しきれないほど大きな数です。

このような巨大数について、2ちゃんねる掲示板というインターネットの掲示板でなされた議論の一部をまとめます。2ちゃんねる掲示板では、話題のジャンルごとに掲示板群「板」が設定され、板の中で話題ごとに立てられる「スレッド」において、会話がなされます。数学の話題を扱う「数学板」の中の「巨大数探索スレッド」シリーズの中で、巨大数探索が繰り返されてきました。

この巨大数探索スレッドに「ふいっしゅっしゅ」という名前で書き込んでいるのが、この文章の著者です。「ふいっしゅっしゅ」とはふざけた名前ですが、当初スレッドに参加したときには、一時的なお遊び程度に思っていたためです。まさか、ここまでまじめに白熱した議論が戦わされるとは思っていませんでした。

第2章 グラハム数を超えて

2.1 グラハム数

グラハム数は、非常に大きい数です。まともに考えると、気の遠くなるほど大きな数です。「巨大数探索スレッド」のきっかけとなった「グラハム数スレッド」では、以下の様に紹介されています。

4 名前：1 3 2 人目の素数さん : 02/02/18 20:52

東京書籍の「数の事典」によると

3 の 3 乗を 3^3 と表記する。 $3^3 = 27$

$3^{3^3} = 3^{(3^3)} = 3^{27} = 3$ の 27 乗

$3^{3^{3^3}} = 3^{(3^{3^3})} = 3^{(3^{27})}$ (3^3 の 27 乗)

つまり矢印 3 個でもう、とんでもないでかい数になる (3 の 10 兆乗の 10 兆乗くらいか?)

さらに $3^{3^{3^{3^3}}}$ なんて、もう宇宙にあるものすべてを使っても 3 の 乗 という表記すらできないくらいでかい。

次はその数 ($3^{3^{3^{3^3}}}$) だけの矢印が 3 の間に挟まった数、次はその数だけ 3 の間に矢印が挟まった数 と繰り返して行って 63 段階目にグラハム数に到達できるそうだ。

気が狂いそうदार。

ここで、 \rightarrow は Knuth の矢印表記 (arrow notation) あるいはタワー関数 (power tower) と言われ、以下で定義されます。

【定義】 Knuth の矢印表記

$$x \rightarrow y = x^y$$

$$x \rightarrow 2 = x^x, \quad x \rightarrow x = x^{(x \rightarrow (x \rightarrow (y - 1)))}$$

$$x \rightarrow 1 = x, \quad x \rightarrow x = x^{(x \rightarrow (x \rightarrow (y - 1)))}$$

$$x \rightarrow 0 = 1, \quad x \rightarrow x = x^{(x \rightarrow (x \rightarrow (y - 1)))}$$

この矢印表記を元に、グラハム数 (Graham's number) は以下のように定義できます。

【定義】 グラハム数

$$f(x) = 3 \rightarrow \dots (x \text{ 個}) \dots \rightarrow 3 \text{ としたときの } f^{64}(4) \text{ を、グラハム数とする。}$$

このグラハム数という数がどのくらい大きいか、想像力をたくましくした結果、以下の様に記述されています。

2.2 ふいつしゅ数

ふいつしゅ数 (Fish number) は、当初グラハム数よりも大きな数を定義する目的で作られました。もちろん、グラハム数を使って「グラハム数+1」「グラハム数のグラハム数乗」などとすればグラハム数よりも大きな数はできますが、

1. グラハム数そのものを定義に用いない
2. グラハム数の定義とは異なったアプローチをする

といった条件を満たさなければ、そこに面白味はありません。

これがふいつしゅ数の定義です。

【定義】ふいつしゅ数バージョン 1

[1] 自然数と関数のペアから、自然数と関数のペアへの写像 S (S 変換) を以下で定義する。

$$S: [m, f(x)] \quad [g(m), g(x)]$$

ただし $g(x)$ は以下で与えられる。

$$B(0, n) = f(n)$$

$$B(m+1, 0) = B(m, 1)$$

$$B(m+1, n+1) = B(m, B(m+1, n))$$

$$g(x) = B(x, x)$$

[2] 自然数、関数、 S 変換から同様の組を生み出す写像 SS を以下で定義する。

$$SS: [m, f(x), S] \quad [n, g(x), S2]$$

ただし $S2$ と $n, g(x)$ は順次以下で与えられる。

$$S2 = S^{f(m)}$$

$$S2 : [m, f(x)] \quad [n, g(x)]$$

[3] $[3, x+1, S]$ に SS 変換を 63 回繰り返して得た自然数をふいつしゅ数、関数をふいつしゅ関数とする。

このふいつしゅ数の大きさは、以下の様になります。

1. $[3, x+1]$ に S 変換を 1 回繰り返すと、アッカーマン関数 (次章参照) 程度の大きさの関数ができる。
2. $[3, x+1]$ に S 変換を 2 回繰り返すと、グラハム数よりも大きな数ができる。

巨大数探索スレッドでは、このふいつしゅ数を具体的に計算しようとして、それがいかに巨大な数になるか、といったことで盛り上がりを見せます。それが、巨大数探求の道へのスタートでした。

ここから先、色々なバージョンのふいつしゅ数が登場することになるため、このふいつしゅ数を「ふいつしゅ数バージョン1」と名付けます。また、ふいつしゅ数バージョン1を F_1 、ふいつしゅ関数バージョン1を $F_1(x)$ とします。一般に、ふいつしゅ数バージョンNを F_N 、ふいつしゅ関数バージョンNを $F_N(x)$ と表記します。

なぜ、このふいつしゅ数がグラハム数よりも大きくなるのか、そして、さらに巨大な数を作るためにはどのようにすれば良いのか、次章以降で探求していきます。

2.3 ふいつしゅ数誕生の経緯

次章以降では、専門的な話に入って行きますので、その前に、ふいつしゅ数誕生の経緯について記しておきます。本節の内容のより数学的な説明は、次章以降で整理した形で記述されますので、本章を飛ばして次章に進んでいただいて構いません。

そもそものきっかけは、大きな数を作ろう!ということでした。たとえば、グラハム数を使って以下の様な数が定義されました。

161 名前：1 3 2 人目の素数さん : 02/06/20 22:25

>> 156 ジャアグラハム数でいってみよう

グラハム数の定義をご存知だと思うが $3 \uparrow 3$ (これがどれだけ超巨大かはグラハム数スレ参照) の数だけ 3 と 3 の間に 3 が挟まった数を1段階として、2段階は1段階の数だけ 3 と 3 の間に 3 がある数と繰り返した63段階目の数がグラハム数と定義される。

この前段階の数だけ 3 が挟まる数が次の段階という63回の変換の1回をG変換と名付ける。

- N 0 1 グラハム数回だけG変換した数回変換した数回変換した~この繰り返しを
- N 0 2 グラハム数回だけG変換した数回変換した数回変換した~この繰り返しを
- N 0 3 グラハム数回だけG変換した数回変換した数回変換した~この繰り返しを
- N 0 4 グラハム数回だけG変換した数回変換した数回変換した~この繰り返しを

とやっていって、N o がグラハム数回まで到達したら終り

ただだと面白くないので

このN o の繰り返しをグラハム数回のG変換した数回変換した数回変換した
~この繰り返しをグラハム数回のG変換した数回変換した数回変換した
上と同じく1行目がN 01、2行目がN 02としてN 0 グラハム数までいって終了

ただだと面白くないので

このN 0 の繰り返しをグラハム数回のG変換した数回変換した数回変換した
~この繰り返しをグラハム数回のG変換した数回変換した数回変換した
上と同じく1行目がN O 1、2行目がN O 2としてN O グラハム数までいって終了

以上のように延々繰り返してN O の種類がグラハム数種類に到達した時の数

これに対して、このように大きな数を定義するプロセスを追う事で、グラハム数を定義の中で用いずに、さらにグラハム数よりも大きな数、上記の数よりも大きな数を定義しよう、という目的で、以下の様な考察がされました (同スレッドの 317-319 の発言)。

これまでの書き込みで「いかにして大きな数を作るか」というプロセスを一般化すると、大きな数と増加の程度が大きい関数を生み出していくプロセスだと表現できる。

たとえば、「 m という数に $f(x)$ という変換を n 回繰り返す」という表現をするときに、 $m, f(x), n$ に使えるのは今までに定義された数と関数のみ。そこで、数と関数を双方ともに帰納的に定義していくプロセスを追っていくことにする。

そこで、自然数 m と関数 $f(x)$ のペアから、自然数 n と関数 $g(x)$ のペアを生み出す変換 (写像) を

$$S: [m, f(x)] \quad [n, g(x)]$$

と表記することになると、たとえば $3 \quad 3$ は、自然数 4 と、 $f(x) = 3$ (3 が x 個) 3 から $f(4)$ とあらわされる。 $3 \quad 3$ 個だけ 3 がはさまった数、は $f(f(4))$ である。したがって、これを 64 回繰り返した数は $f^{64}(4)$ となり、この操作は $g(x) = f^{64}(x)$ という関数を自然数 64 と関数 $f(x)$ から生み出す操作にほかならないため、

$$S: [m, f(x)] \quad [f^m(m), f^m(x)]$$

と書くと、 $m=64, f(x)$ からグラハム数よりも大きい $f^{64}(64)$ という数が生み出される。

これまでのスレッドにかかれた数は、いろいろなタイプの S 変換を数回、 $\gg 161$ でもせいぜい 10 回程度行っているにすぎない。 S 変換については、上記の S 変換よりも、Ackermann タイプの S 変換の方がより関数を増加させる。

そこで、これから先は「いかにしてより大きな数、関数を生み出す S 変換を作り出すか (これを「より大きい S 変換」と呼ぶ)」といった考察をする。

その第 1 段階として、Ackermann 関数にならい、

$$B(0, n) = f(n)$$

$$B(m+1, 0) = B(m, 1)$$

$$B(m+1, n+1) = B(m, B(m+1, n))$$

$$g(x) = B(x, x)$$

としたときに、 $S: [m, f(x)] \quad [g(m), g(x)]$ とする。少なくとも、これ以上に大きい S 変換はこれまでにあらわれていない。したがって、たとえば $[3, f(x) = x+1]$ にこの S 変換を 10 回ほど繰り返せば、ゆうに $\gg 161$ を越える。

では、この S 変換をさらに大きくするにはどうすれば良いか。

それには、S 変換を $f(m)$ 回繰り返した変換を S2 変換とすれば良い。すなわち、 $m, f(x), S$ からさらに大きな S2 変換を生み出すことができる。このプロセスを、

$$SS:[m, f(x), S] \quad [n, g(x), S2]$$

$$\text{ただし } g(x) = S2[m, f(x)], n = g(m)$$

という SS 変換で記述することにする。

$[3, x+1, S]$ に SS 変換を 1 回かけると、S 変換を 4 回くりかえす変換が得られ、さらにもう 1 回かけると、S 変換を大変な数繰り返した変換が得られるため、2 回くりかえしただけで、すでにこのスレッドに登場したいかなる有限の数よりも大きな数が得られる。

この発言の直後に、ふいつしゅ数が定義されています。このように、数と関数から数と関数を生成する「S 変換」という概念を導入する事で、巨大数を生み出そうとしたのがふいつしゅ数です。このことは、次章以降で関数から関数を生成する「作用素」「操作」という概念で、より明確に説明されます。

その後、ふいつしゅ数の大きさが計算されました。同スレッドの 329, 331, 377-379 の計算内容を転記します。

$[3, f(x)=x+1]$ に S 変換を 1 回すると、

$$B(0, n) = n + 1$$

$$B(m+1, 0) = B(m, 1)$$

$$B(m+1, n+1) = B(m, B(m+1, n))$$

$$g(x) = B(x, x)$$

となるので、 $B(m, n)$ はアッカーマン関数と一致し、

$g(x) = A(x, x)$ となるため、

$$S:[3, x+1] \quad [A(3, 3), A(x, x)]$$

となる。

$A(3, 3) = 61$ なので、S 変換 1 回ではまだたいした大きさにはならない。

S 変換の 2 回目。今度は、

$$B(0, n) = A(n, n)$$

$$B(m+1, 0) = B(m, 1)$$

$$B(m+1, n+1) = B(m, B(m+1, n))$$

$$g(x) = B(x, x)$$

となるが、この $g(x)$ 関数はとてつもない関数になる。

$$g(1) = B(1, 1) = B(0, B(1, 0)) = B(0, B(0, 1)) = B(0, A(1, 1))$$

$$= B(0, 3) = A(3, 3) = 61$$

$$g(2) = B(2, 2) = B(1, B(2, 1)) = B(1, B(1, B(2, 0)))$$

$$= B(1, B(1, B(1, 1))) = B(1, B(1, 61))$$

$$= B(1, B(0, B(1, 60)))$$

このあたりで、すでに書き下すことが困難になってくる。

$$B(1,1)=61$$

$$B(1,2)=A(61,61)$$

$$B(1,3)=A(A(61,61),A(61,61))$$

という調子で関数が増えていくので、 $B(1,61)$ はとんでもない数。 $g(2)=B(1,B(1,61))$ なので、 $g(2)$ ですでにグラハム数を超えているように思う。

$g(2)$ ですでにグラハム数を超えてしまい、さらに $g(x)$ は x が増えるにつれてものすごい勢いで増えるので、 $g(61)$ の大きさは想像を絶する。S変換 2 回目にして、 $g(61)$ というとんでもない数が得られることになる。

S変換の3回目は

$$B(0,n)=g(n)$$

$$B(m+1,0)=B(m, 1)$$

$$B(m+1,n+1)=B(m, B(m+1, n))$$

$$gg(x)=B(x,x)$$

としたときの $gg(x)$ となるので、

$$gg(1)=B(1,1)=B(0,B(1,0))=B(0,B(0,1))=B(0,g(1))$$

$$=B(0,61)=g(61)$$

$$gg(2)=B(2,2)=B(1,B(2,1))=B(1,B(1,B(2,0)))$$

$$=B(1,B(1,B(1,1)))=B(1,g(61))$$

$$B(1,1)=g(61)$$

$$B(1,2)=g(g(61))$$

$$B(1,3)=g(g(g(61)))$$

つまり、 $gg(2)$ は 61 を $g(x)$ に代入して...と $g(61)$ 回繰り返した数。この調子で $gg(3), gg(4)...$ と増えていき、 $gg(g(61))$ が S 変換を 3 回繰り返した数。

SS 変換 2 回目は、SS 変換 1 回によって得られた $m, f(x), S$ に対して $S^{f(m)}$ とする、すなわち S 変換 (つまり最初の S 変換を 4 回繰り返す変換) を $f(m)$ 回繰り返す。ここで、 $f(m)$ 回とは SS 変換 1 回、つまり S 変換 4 回によって得られる大きな数 m を、これまた S 変換 4 回によって得られる増加率の大きな関数 $f(x)$ に代入した数なので、とてつもなく大きな数。その数だけ、新 S 変換を繰り返す、ということ。

つまり、大きな数と関数から大きな数と関数を生み出し、その生み出された大きな数と関数から大きな S 変換を生み出し、大きな S 変換がさらにとてつもなく大きな数と関数を生み出す、とお互いがお互いを増幅させていく。

この計算に対して、巨大数探索スレッドでは以下の様な反応となりました。

380 名前：1 3 2 人目の素数さん : 02/07/02 19:47

うぎゃ-----っ!!!

でっ、でけえ!!!

フィッシュ数は文句なし世界一、宇宙一の数だ!!!!!!

グラハム数とフィッシュ数を比べると、グラハム数は限りなく0に近い

お疲れ様でした、

386 名前：1 3 2 人目の素数さん : 02/07/02 20:25

ていうか

現実にあるものを文字にするだけの数字とかより

フィッシュ数のこと考える!

もうすごいから。すごいしヤバイから。

388 名前：1 3 2 人目の素数さん : 02/07/02 20:32

小・中学生の頃、講談社のブルーバックスの宇宙や物理の本で

全宇宙のニュートリノの数とか光子の数が 10^{88} とかいう数字を見て

ぶったまげてたのが、なんかカワイク思えるよ

単に「とても大きな数を定義した」という以上の何者でもないのですが、これがこのように感銘を呼ぶところが巨大数の不思議な魅力です。

このふいつしゅ数は、巨大数探索の第一歩に過ぎません。これがきっかけとなり、さらに大きな巨大数を作るにはどうすればいいのか、この巨大数とこの巨大数は、どちらの方が大きいのか、といった議論が続く事になります。議論には、数学の専門家も加わるようになり、関連する数学の内容を学びながら、巨大数の探索が続きました。そこには、この「ふいつしゅ数バージョン1」がとても小さい数に思える様な、果てしなく巨大な数の世界がありました。また、巨大数の議論をきっかけとして、高度な数学的概念を学ぶ機会となったことは、有益でした。

その議論の内容を、次章以降で整理していきたいと思います。

第3章 2重帰納関数

3.1 巨大数と巨大関数

巨大数を作るということは、巨大関数を作るということに他なりません。巨大関数を作るとは、つまり増加率の高い関数を作ることです。

四則演算しか知らない人が、巨大数を作るとしたら、たとえば「 99999×99999 」とか「 $999999 \times 999999 \times 999999$ 」などを書くでしょう。ところが、このようにしてたくさん9を並べて、 \times をいっぱい使って数を書いたところで、「 $10^{10^{1000}}$ 」にはかないません。「 $10^{10^{1000}}$ 」ほどの数を作るためには、 10^{1000} 桁ほどの数字やかけ算記号を書く必要があるからです。これは、四則演算によって作られる関数よりも、べき乗関数の方が増加率が飛躍的に高いためです。また、グラハム数の定義では「 n を n 個並べる数」という巨大関数が定義されたことで、四則演算やべき乗だけではとても表記しきれないほど大きな巨大数が定義されています。

このように、より増加率の高い関数を定義することで、より巨大な数を作ることができるため、巨大数を作ることは巨大関数を作る事と同じことになります。今後は、いかにして巨大関数を作るか、という議論が中心になります。それはそのまま、巨大数を作る、ということになります。したがって、本書のタイトルも「巨大関数論」とする方が内容に即していますが、巨大数への想いを込めて、あえて「巨大数論」としました。

まとめると、

1. 本書で扱う巨大数は非常に大きな有限の自然数である。
2. 巨大数は、増加率の高い巨大関数によって作られる。ここで、巨大関数は定義域と値域が非負整数である非負整数個の引数をとる単調増加関数であるとする。

3.2 アッカーマン関数と原始帰納関数

3.2.1 アッカーマン関数

本節では、非常に増加率が高いことで有名なアッカーマン関数を紹介し、続いて、なぜアッカーマン関数の増加率が我々が通常目にする関数と比べて飛躍的に増加率が高いか考察をします。

【定義】アッカーマン関数

非負整数 x, y に対し、以下のように定義される関数 $A(x, y)$ をアッカーマン関数 (Ackermann's Function) とする。

$$A(0, y) = y + 1 \tag{3.1}$$

$$A(x + 1, 0) = A(x, 1) \tag{3.2}$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y)) \tag{3.3}$$

$A(x, y)$ は、非常に増加率の大きい関数です。具体的に計算してみましょう。

$$A(1, 1) = A(0, A(1, 0)) = A(0, A(0, 1)) = A(0, 2) = 3$$

$$A(1, 2) = A(0, A(1, 1)) = A(0, 3) = 4$$

$$A(1, y) = A(0, A(1, y - 1)) = A(1, y - 1) + 1 = y + 2$$

$$A(2, 1) = A(1, A(2, 0)) = A(1, 3) = 5$$

$$A(2, y) = A(1, A(2, y - 1)) = A(2, y - 1) + 2 = 2y + 3$$

$$A(3, 1) = A(2, A(3, 0)) = A(2, 5) = 13$$

$$A(3, 2) = A(2, A(3, 1)) = A(2, 13) = 29$$

$$A(3, 3) = A(2, 29) = 61$$

$$A(3, y) = A(2, A(3, y - 1)) = 2 * A(3, y - 1) + 3 > 2^y$$

$$A(4, 1) = A(3, 13) > 2^{13}$$

$$A(4, 2) > A(3, 2^{13}) > 2^{2^{13}}$$

$$A(4, y) > 2^{2^{2^{2^{\dots(y \text{ 個})}}}} \dots ^2$$

$$A(5, y) = A(4, A(5, y - 1)) > 2^{2^{2^{2^{\dots(A(5, y - 1) \text{ 個})}}}} \dots ^2$$

このように、 $x=1, 2$ の時は y に対して線形に増加し、 $x=3$ の時は y に対して指数的に増加します。 $x=4$ の時は、「指数をいくつつなげるか」という数が増加する程度に y とともに増加し、 $x=5$ の時は、さらに大きな増加率を示します。こうして、 x の値が大きくなるにつれて、 y の値に対する $A(x, y)$ の増加率が飛躍的に大きくなります。

3.2.2 原始帰納関数

このアッカーマン関数の増加率の大きさを理解するためには、まず「原始帰納関数」について理解する必要があります。

原始帰納関数（あるいは原始再帰関数; Primitive recursive function）とは、合成と原始帰納で定義される関数で、以下のように定義されます。

【定義】原始帰納関数

原始帰納関数とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、

1. 定数関数: $f(x)=0$
2. 後者関数: $f(x)=x+1$
3. 射影関数: 複数の引数を持つ関数から、 i 番目の引数を返す関数
4. 関数の合成: f と g の合成関数 $h: h(x_1, \dots, x_m) = f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$
5. 原始帰納: f と g の原始帰納関数 $h: h(0, x_1, \dots, x_k) = f(x_1, \dots, x_k), h(n+1, x_1, \dots, x_k) = g(h(n, x_1, \dots, x_k), n, x_1, \dots, x_k)$

以上の作用素（操作）を有限回適用した関数である。

たとえば、加算、乗算、べき乗は、原始帰納的に定義できるため、それらのみを有限回繰り返した関数は、原始帰納関数です。

3.2.3 2重帰納関数

さて、アッカーマン関数はどんな原始帰納関数よりも増加が強い関数です。つまり、任意の原始帰納関数 $f(x)$ に対して、

$$f(x) < A(c, x) \tag{3.4}$$

となる c が存在します。このことを、以下で示します。なお、ここでは1変数関数について議論を進めますが、 $f(x)$ が多変数関数であっても同様の議論ができます。

原始帰納の定義式において $B(x) = h(n, x), C(y) = g(y, n, x)$ と書くと、

$$B(x+1) = C(B(x)) \tag{3.5}$$

$$B(x) = C^x(B(0)) \tag{3.6}$$

と書けます。つまり、 $B(x)$ は関数 $B(0)$ に対して $C(y)$ の合成を x 回繰り返した関数です。合成の回数を変数とする関数を作ることが、原始帰納の操作となっています。このことを「操作を数え上げる」と表記することとします。すなわち、原始帰納の操作は合成操作を数え上げます。したがって、関数 $f(x)$ から、合成操作を有限回 (n 回) 繰り返して得られるいかなる関数 $g(x)$ よりも、初期関数から原始帰納の操作によって得られる関数 $h(x)$ の方が大きくなります。たとえば、 $f(x)$ に $f(f(x))$ の合成操作を n 回繰り返して得られる関数を $g_n(x)$ として、

$$g_1(x) = f^2(x) : g_1(1) = f^2(1), g_1(2) = f^2(2), \dots, g_1(n) = f^2(n) \tag{3.7}$$

$$g_2(x) = f^3(x) : g_2(1) = f^3(1), g_2(2) = f^3(2), \dots, g_2(n) = f^3(n) \tag{3.8}$$

...

$$g_n(x) = f^{n+1}(x) : g_n(1) = f^{n+1}(1), g_n(2) = f^{n+1}(2), \dots, g_n(n) = f^{n+1}(n) \tag{3.9}$$

といった表を作ります。このとき、関数 $f(x)$ に合成操作を x 回繰り返して得られる関数 $g_x(x) = f^{x+1}(x)$ は、

$g_x(x) = f^{x+1}(x) : g_x(1) = f^2(1), g_x(2) = f^3(2), \dots, g_x(n) = f^{n+1}(n)$ と書く事ができて、これは上の表において右辺を左上から右下へ対角線上に拾い上げています。

このように、操作の回数を数え上げることは対角線上に拾い上げて関数を作成する操作であることから、「操作の対角化」という言葉が巨大数探索スレッドの中で使われています。これは、いわゆるカントールの対角線論法と言われる論法です。

さて、アッカーマン関数の漸化式 (3.3) において、 $f(y) = A(x, y), g(y) = A(x+1, y)$ とすると、 $g(y+1) = f(g(y))$ となり、したがって、 $g(y) = f^y(g(0)) = f^y(f(1)) = f^{y+1}(1)$ と書けます。すなわち、 $g(y)$ は関数 $f(y)$ に対して合成を y 回繰り返して 1 を代入した関数です。これは対角線ではありませんが関数の合成回数を数え上げています。関数 $f^n(y)$ と $f^{(y+1)}(1)$ を比べると、 $y \geq 10n$ において

$$f^{y+1}(1) = f^n[f^{y-n+1}(1)] > f^n[f^{0.9y+1}(1)] > f^n(y)$$

となるため、合成を有限回繰り返したいかなる関数よりも合成回数を数え上げる関数の方が大きくなります。つまり、合成回数を数え上げることで、関数の対角列を取る「操作の対角化」と等しい効果が得られます。

このように、 $f(y)$ から $g(y)$ を生成する操作は、合成を数え上げる原始帰納操作となるため、 $A(x, y)$ に立ち戻って考えると、 $A(x+1, y)$ は $A(x, y)$ に対して原始帰納の操作を 1 回操作した関数です。このことから、 $A(x, y)$ は $A(0, y)$ に対して原始帰納の操作を x 回繰り返した関数であり、原始帰納の操作を数え上げています。このように、原始帰納を数え上げる 2 重帰納を、以下で定義します。

【定義】2 重帰納

原始帰納操作を数え上げる操作を 2 重帰納作用素（操作）とする。

2 重帰納関数 (double recursive function) とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、定数、後者、射影、合成、原始帰納、2 重帰納の作用素（操作）を有限回適用した関数である。ただし、その中で 2 重帰納操作を少なくとも 1 回含むものとする。

射影、合成、原始帰納の操作を n 回繰り返して作られた原始帰納関数 $f(x)$ について、その中で最も関数の増加度が大きい原始帰納の操作を n 回繰り返して作られた関数 $A(n, y)$ は大きくなります。すなわち、 $x > n$ において $f(x) > A(x, y)$ ですから、 $A(x, y)$ は $f(x)$ と比べて本質的に大きい、ということになります。

3.2.4 グラム数とふいつしゅ数

グラム数がなぜ気が遠くなるほど大きい数となるか、あらためてグラム数の定義を見ましょう。 $f(x, y) = 3 \dots (x \text{ 個}) \dots y$ と書くと、 $f(x+1, y) = f(x, f(x, y))$ と定義され、これは 2 重帰納関数です。この 2 重帰納関数に対して、 $f^{64}(4, 3)$ がグラム数です (合成の際には、 $f(x, y)$ の x に代入をする)。すなわち、グラム数は原始帰納関数に対して 1 回の 2 重帰納と 63 回の合成を施した関数 $f^{64}(x)$ を作成して、そこに数を代入した、と見る事ができます。

一方、ふいつしゅ数 F_1 の S 変換は、関数 $f(x)$ に 2 重帰納の操作をして関数 $g(x)$ を生成する操作です。 F_1 は、関数に 2 重帰納の操作を数多く繰り返すことによって、より大きな関数を生成するしくみである、と表現できます。したがって、2 重帰納を 2 回施した時点で、2 重帰納を 1 回しか施していないグラム数と比べて比較にならないほど大きな関数が生成されることとなります。ただし、この議論は厳密ではありません。なぜならば、2 重帰納の操作は何度繰り返しても 2 重帰納のままなので、2 重帰納の回数では生成され

る関数の大小を比較できないからです。ここでは、グラハム数生成に使われている操作も F_1 の S 変換も同等のアッカーマン的な 2 重帰納操作であるため、このような比較が可能となっています。

SS 変換は、S 変換を数多く繰り返す操作です。その繰り返し回数は、それまでに作られた 2 重帰納関数から作られた数に、それまでに作られた関数を代入したものですから、2 重帰納関数によって作られる数の回数だけ繰り返す操作です。2 重帰納を繰り返す回数とはとても大きくなりますが、あくまでも S 変換を 1 定数繰り返す操作となるため、2 重帰納の範囲を出ることはありません。したがって、SS 変換は 2 重帰納操作から、より大きな 2 重帰納操作を作り出す変換ということになります。

— 本節のまとめ —

1. 操作の一定数回の繰り返しよりも、操作の数え上げの方が強い関数を生成する。
2. 合成操作を数え上げると原始帰納となる。したがって、原始帰納の操作は合成操作よりも強い。
3. 原始帰納操作を数え上げると 2 重帰納となる。したがって、2 重帰納の操作は原始帰納操作よりも強い。
4. アッカーマン関数は 2 重帰納関数である。
5. F_1 の S 変換と SS 変換は、2 重帰納操作である。したがって、 $F_1(x)$ は 2 重帰納関数である。

3.3 チェーン表記

3.3.1 定義と計算

矢印表記を一般化した Conway の矢印チェーン表記 (Conway's chained arrow notation) の定義を、以下に記します。

— 【定義】 Conway のチェーン表記 —

基本的な定義 (3 つの数字が並んでいる場合) は以下のとおりです。

$$a \dots (c \text{ 個}) \dots b = a \quad b \quad c$$

さらに、以下のように、4 つ以上の数字を連ねて書くことができます。

チェーンの最後の数が 1 のときはこれを落とすことができます。

$$a \quad b \quad \dots \quad x \quad y \quad 1 = a \quad b \quad \dots \quad x \quad y$$

チェーンの最後から 2 番目の数が 1 の場合、これと最後の数をまとめて落とせます。

$$a \quad b \quad \dots \quad x \quad 1 \quad z = a \quad b \quad \dots \quad x$$

次のような変形によって最後とその前の数を減らすことができます。

$$a \quad b \quad \dots \quad x \quad y \quad z = a \quad b \quad \dots \quad x \quad (a \quad b \quad \dots \quad x \quad y-1 \quad z) \quad z-1$$

このチェーンは、とても増加率の高い関数です。具体的に、いくつか値を計算してみましょう。

$$10 \quad 2 \quad 2 = 10 \quad (10 \quad 1 \quad 2) \quad 1 = 10 \quad 10 = 10^{10}$$

$$10 \quad 3 \quad 2 = 10 \quad (10 \quad 2 \quad 2) \quad 1 = 10 \quad (10 \quad 2 \quad 2) = 10^{10^{10}}$$

この時点で、無量大数 (10^{68})、エディントン数 (10^{79}) といった数を超えます。

$$10 \quad 4 \quad 2 = 10 \quad (10 \quad 3 \quad 2) \quad 1 = 10^{10^{10^{10}}}$$

この時点で、不仮説不仮説転 ($10^{3.72 \times 10^{37}}$) を超えます。

a b 2 は $a^{a^{\dots^a}}$ と a のべき乗を b-1 回繰り返した数です。したがって、

$$3 \quad 2 \quad 3 = 3 \quad (3 \quad 1 \quad 3) \quad 2 = 3 \quad 3 \quad 2 = 3^3 = 3^{27}$$

$$3 \quad 3 \quad 3 = 3 \quad (3 \quad 2 \quad 3) \quad 2 = 3 \quad 3^{27} \quad 2 = 3 \text{ のべき乗を } 3^{27} - 1 \text{ 回繰り返した数}$$

となります。

グラハム数は、 $f(x) = 3 \quad \dots(x \text{ 個})\dots \quad 3 = 3 \quad 3 \quad x$ としたときの $f^{64}(4)$ です。

$$3 \quad 3 \quad 1 \quad 2 = 3 \quad 3 \quad 1 = f(1) = 27$$

$$3 \quad 3 \quad 2 \quad 2 = 3 \quad 3 \quad (3 \quad 3 \quad 1) = f^2(1) = f(27)$$

$$3 \quad 3 \quad 3 \quad 2 = 3 \quad 3 \quad (3 \quad 3 \quad 2 \quad 2) = f^3(1) = f^2(27)$$

$$3 \quad 3 \quad n \quad 2 = f^n(1) = f^{n-1}(27)$$

$$3 \quad 3 \quad 64 \quad 2 = f^{64}(1) < f^{64}(4) = \text{グラハム数}$$

$$3 \quad 3 \quad 65 \quad 2 = f^{64}(27) > f^{64}(4) = \text{グラハム数}$$

したがって、

$$3 \quad 3 \quad 64 \quad 2 < \text{グラハム数} < 3 \quad 3 \quad 65 \quad 2$$

となります。さらに、

$$3 \quad 3 \quad 3 \quad 3 = 3 \quad 3 \quad (3 \quad 3 \quad 2 \quad 3) \quad 2 > 3 \quad 3 \quad 65 \quad 2$$

となります。

以上をまとめます。

チェーンによる巨大数の比較

$$\text{無量大数} < \text{エディントン数} < 10 \quad 3 \quad 2 < \text{不可説不可説転} < 10 \quad 5 \quad 2 < 3 \quad 3 \quad 3 < 3 \quad 3 \quad 64 \quad 2 < \text{グラハム数} < 3 \quad 3 \quad 65 \quad 2 < 3 \quad 3 \quad 3 \quad 3$$

このように、チェーンの最後の数を増やすこと、そしてチェーンそのものを伸ばすことにより、増加度が加速的に増す関数がチェーン表記です。

3.3.2 2重帰納

チェーン表記は、2重帰納関数です。

まず、3変数のチェーンについては

$$a \quad \dots(c \text{ 個})\dots \quad b = a \quad b \quad c$$

と定義されますので、タワーの定義式

$$x \quad y = x \quad (x \quad (y - 1))$$

をチェーンで表記すると、

$$x \quad y \quad 3 = x \quad (x \quad (y-1) \quad 3) \quad 2$$

と、多変数の定義式と同じになります。

したがって、チェーンの基本計算式は、多変数のこの式となります。

$$a \quad b \quad \dots \quad x \quad y \quad z = a \quad b \quad \dots \quad x \quad (a \quad b \quad \dots \quad x \quad y-1 \quad z) \quad z-1$$

ここで、アッカーマン関数

$$A(x+1,y+1) = A(x,A(x+1,y))$$

について、 $x+1 = z$, $y+1 = y$ として変数の順番を交換し、

$$(y,z) = A(A(y-1,z), z-1)$$

さらに定数項 a,b,\dots,x を加えると

$$A(a,b,\dots,x,y,z) = A(a,b,\dots,x,A(y-1,z), z-1)$$

となってチェーンの式と一致することから、チェーンの漸化式はアッカーマンと同じ2重帰納の式です。したがって、チェーン関数は2重帰納をたくさん繰り返す2重帰納関数である、ということになります。

3.3.3 F_1 とチェーン表記の比較

チェーン表記、 $F_1(x)$ ともに2重帰納関数です。それぞれ、アッカーマン関数を元のような原始帰納を数え上げる基本2重操作を持っています。チェーンでは、矢印を1つ伸ばすこと、すなわち、変数を1つ増やすことで、基本2重帰納操作を1回することになります。変数が n 個のチェーンでは、基本2重帰納操作が $n-2$ 回行われます。 $F_1(x)$ では、初期のS変換を1回施す事が、基本2重帰納操作を1回することになります。グラハム数の大きさを見ると、それは基本2重帰納操作を1回だけしかしていないため、2回行った4変数チェーン(3 3 3 3)、あるいはS変換を2回繰り返すことで、グラハム数よりも大きな数を得る事ができます。

$F_1(x)$ の大きさは、基本2重帰納操作を施す回数が大変大きい、というところにあります。したがって、チェーン表記によって「9 9 9 9 9 9 9 9 ...」といった表記を現実的に紙に書けるほどの回数したところで、 F_1 よりも大きい数を得ることはできません。また、 F_1 の定義の中には、2重帰納操作の回数を2重帰納関数によって得られた回数繰り返す、といった操作もまた含まれているため、たとえば「9 9 ... (9 9 9回) 9」といった数よりも、 F_1 の方が大きくなります。このように、 $F_1(x)$ は2重帰納関数の中では非常に大きな関数です。同じ2重帰納というカテゴリの中では、常識的な長さのチェーンよりも $F_1(x)$ の方が大きい、という表現もできると思います。しかし、結局のところ、チェーンの長さを F_1 程度にした関数を作れば、それは $F_1(x)$ よりも大きくなりますから、本質的にはチェーンも $F_1(x)$ も同じ程度の大きさを持つ2重帰納関数である、ということになります。

第4章 3重帰納関数

4.1 3変数アッカーマン関数

前章で、2重帰納関数である2変数アッカーマン関数、ふいっしゅ関数バージョン1、チェーン表記の大きさを見ました。この章では、3重帰納関数の大きさを見ることとします。2重帰納関数が、いかなる原始帰納関数よりも強い関数であることと同様に、3重帰納関数は、いかなる2重帰納関数よりも強い関数です。

【定義】3重帰納

2重帰納操作を数え上げる操作を3重帰納作用素（操作）とする。

3重帰納関数 (triple recursive function) とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、定数、後者、射影、合成、原始帰納、2重帰納、3重帰納の作用素（操作）を有限回適用した関数である。ただし、その中で3重帰納操作を少なくとも1回含むものとする。

まずは、3重帰納関数の例として、2変数アッカーマン関数を拡張して3変数アッカーマン関数を以下の様に定義します。

【定義】3変数アッカーマン関数

非負整数 x, y, z に対し、以下のように定義される関数 $A(x, y, z)$ を3変数アッカーマン関数とする。

$$A(0, 0, z) = z + 1 \quad (4.1)$$

$$A(x + 1, 0, z) = A(x, z, z) \quad (4.2)$$

$$A(x, y + 1, 0) = A(x, y, 1) \quad (4.3)$$

$$A(x, y + 1, z + 1) = A(x, y, A(x, y + 1, z)) \quad (4.4)$$

この3変数アッカーマン関数が、2重帰納を数え上げる3重帰納操作を含み、したがってどんな2重帰納関数よりも大きい3重帰納関数であることを示します。

2変数アッカーマンの定義と見比べることにより、 $A(0, y, z) = A(y, z)$ が成り立ちます。また、 $A_x(y, z) = A(x, y, z)$ とすると、

$$A_x(0, z) = A_{x-1}(z, z) \quad (4.5)$$

$$A_x(y + 1, 0) = A_x(y, 1) \quad (4.6)$$

$$A_x(y + 1, z + 1) = A_x(y, A_x(y + 1, z)) \quad (4.7)$$

となり、 $A_{x-1}(z, z)$ から $A_x(z, z)$ を生成する操作は、2重帰納操作となります。

したがって、 $A(x, y, z) = A_x(y, z)$ は、関数 $A(0, 0, z) = z + 1$ に対して2重帰納操作を x 回繰り返した関数であり、これは2重帰納操作を数え上げる3重帰納関数です。

なお、 F_1 の定義において、 $f(x)$ から $g(x)$ を生成する S 変換の漸化式は、 $f(z) = A_x(z, z)$ 、 $g(z) = A_x(z, z)$ とすると、この 3 変数アッカーマンの式と一致します。このことから、3 変数アッカーマンにおける 2 重帰納操作と F_1 の定義における 2 重帰納操作は、同じであることが分かります。詳しい計算は省略しますが、 F_1 の定義において、 $[3, x+1]$ に S 変換を i 回繰り返して得られた関数を $S_i(x)$ とすると $S_i(x) = A(i, 0, x)$ となり、 F_1 の定義において S 変換の種となる関数を $S_i(x)$ とすると、 $B(m, n) = A(i, m, n)$ となります。前章で、S 変換を 1 回繰り返した後の $g(2)$ がグラハム数を超えるという計算をしましたが、これは、 $A(1, 2, 2)$ がグラハム数よりも大きいことを計算したことになります。

4.2 チェーンの長さを変数化

$(a+1) \dots (a+1) (z+1) (y+1)$ と $a+1$ が x 個続くチェーンを $f_a(x, y, z)$ とします。
チェーンの定義から

$$f_a(0, 0, z) = a^z \quad (4.8)$$

$$f_a(x+1, 0, z) = f_a(x, z, a) \quad (4.9)$$

$$f_a(x+1, y, 0) = f_a(x, 0, a) \quad (4.10)$$

$$f_a(x, y+1, z+1) = f_a(x, y, f_a(x, y+1, z)) \quad (4.11)$$

となります。 $f_a(x, y, z)$ と $A(x, y, z)$ を比べると、若干異なるものの、同程度の増加度の関数となることが分かります。チェーンを 1 個伸ばす (変数の数を増やす) ことは、 $A(x, y, z)$ において x を 1 つ増やすことに相当します。そして、チェーン長が変数化された $f_a(x, y, z)$ は 3 重帰納です。

前章において、 n 変数チェーン表記は、 $n-2$ 回の 2 重帰納操作を含む 2 重帰納関数である事が示されました。そのことと、上記の計算から、 $A(x, y, z)$ は 2 重帰納操作を x 回施した $x+2$ 変数チェーン表記に相当する大きさの関数となります。

4.3 ふいつしゅ数バージョン 2

ふいつしゅ数バージョン 2 は以下の様に定義されます。

【定義】ふいつしゅ数バージョン 2

[1] 自然数と関数のペアから、自然数と関数のペアへの写像 S (S 変換) を以下で定義する。

$$S : [m, f(x)] \quad [g(m), g(x)]$$

ただし $g(x)$ は以下で与えられる。

$$B(0, n) = f(n)$$

$$B(m+1, 0) = B(m, 1)$$

$$B(m+1, n+1) = B(m, B(m+1, n))$$

$$g(x) = B(x, x)$$

[2] 自然数、関数、 S 変換から同様の組を生み出す写像 SS を以下で定義する。

$$SS : [m, f(x), S] \quad [n, g(x), S2]$$

ただし $S2$ と $n, g(x)$ は順次以下で与えられる。

$$S2 = S^{f(m)}$$

$$S2 : [m, f(x)] \quad [n, p(x)]$$

$$S2^x : [m, f(x)] \quad [q(x), g(x)]$$

[3] $[3, x+1, S]$ に SS 変換を 63 回繰り返して得た自然数を F_2 、関数を $F_2(x)$ とする。

F_2 の定義は F_1 と似ていますが、 SS 変換の定義が変わっています。どこが変わっているかという、

1. F_1 の SS 変換は、2 重帰納である S 変換を一定回数繰り返すので 2 重帰納であり、 $F_1(x)$ も 2 重帰納。
2. F_2 の SS 変換は、2 重帰納である S 変換の回数を数え上げるので 3 重帰納であり、 $F_2(x)$ も 3 重帰納。

ということです。どのようにして S 変換を数え上げているのかという、

$$S2^x : [m, f(x)] \quad [q(x), g(x)]$$

この式における、 $f(x)$ と $g(x)$ の関係です。この式において、 S 変換を $f(x)$ 回繰り返した $S2$ 変換を使っていることは、関数を大きくする事に本質的に役立っていないので、

$$S^x : [m, f(x)] \quad [q(x), g(x)]$$

としても同じ事です。この式で生成される $g(x)$ がどんな関数かという (S 変換における関数の変化に着目します)、

$$g(1) \text{ は } f(x) \text{ に } S \text{ 変換を } 1 \text{ 回施して得られる } g_1(x) \text{ に } 1 \text{ を代入した } g_1(1)$$

$$g(2) \text{ は } f(x) \text{ に } S \text{ 変換を } 2 \text{ 回施して得られる } g_2(x) \text{ に } 2 \text{ を代入した } g_2(2)$$

$$g(3) \text{ は } f(x) \text{ に } S \text{ 変換を } 3 \text{ 回施して得られる } g_3(x) \text{ に } 3 \text{ を代入した } g_3(3)$$

$$g(n) \text{ は } f(x) \text{ に } S \text{ 変換を } n \text{ 回施して得られる } g_n(x) \text{ に } n \text{ を代入した } g_n(n)$$

というような関数 $g(x)$ です。この関数 $g(x)$ は、関数 $f(x)$ に S 変換を有限回 (N 回) 施したいかなる関数よりも、 $x > N$ において大きくなります。これが、つまり 2 重帰納である S 変換の操作を数え上げることによって、いかなる 2 重帰納よりも強い 3 重帰納の操作とした S^x (あるいは $S2^x$) を、 SS 変換として定義した、ということです。

S変換を繰り返す回数を定数とするか変数とするかで、生じる関数の大きさが飛躍的に変わり、したがって生成される数の巨大さも飛躍的に変わることになります。そこが F_2 におけるミソでした。

$F_2(x)$ は、3重帰納操作を63回施した3重帰納関数です。

第5章 多重帰納関数

5.1 n変数 Ackermann 関数

【定義】多重帰納

n-1 重帰納操作を数え上げる操作を n 重帰納作用素 (操作) とする。

n 重の多重帰納関数 (multiply recursive function) とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、定数、後者、射影、合成、原始帰納、2 重帰納、3 重帰納、...、n 重帰納の作用素 (操作) を有限回適用した関数である。ただし、その中で n 重帰納操作を少なくとも 1 回含むものとする。

たろう氏によって、アッカーマン関数の拡張 n 変数アッカーマン関数が以下のように定義されました。

【定義】n 変数アッカーマン関数

n 個の非負整数を引数として以下のように定義される関数 A を n 変数アッカーマン関数とする。

$$A(x, 0, a) = a + 1 \quad (5.1)$$

$$A(x, b + 1, 0, a) = A(x, b, a, a) \quad (5.2)$$

$$A(x, b + 1, 0) = A(x, b, 1) \quad (5.3)$$

$$A(x, b + 1, a + 1) = A(x, b, A(x, b + 1, a)) \quad (5.4)$$

ただし、a, b : 0 以上の整数, x : 0 個以上の 0, X : 0 個以上の 0 以上の整数

最後の 2 式により、右から 2 個目の変数は原始帰納、その数え上げが 2 重帰納となります。2 式目が、多重帰納を定義する肝となります。に含まれる 0 の数を n とすると、b の項は右から n+3 番目となります。n=0 の時は、3 変数アッカーマンの式と一致します。つまり、右から 3 変数目の b を 1 つ増やす操作が、右から 2 変数目に変数 a を用いた 2 重帰納の操作となり、右から 3 変数目が 2 重帰納を数え上げる 3 重帰納となります。このようにして、右から n+2 番目の項が n+1 重帰納であり、右から n+3 番目の b の項は、右から n+2 番目の n+1 重帰納を数え上げる (変数 a 回の操作をする) ことで n+2 重帰納の操作となります。したがって、n 変数アッカーマン関数の 1 番左の項は、n-1 重帰納となり、それを変数に持つ n 変数アッカーマン関数は n 重帰納関数となります。

5.2 多重帰納に見えてそうでない関数

2 重帰納も多重帰納の一種ですが、この節に限って 3 重帰納以上を多重帰納とします。チェーンの定義は、一見すると多重帰納に見えますが、2 重帰納の章で示した様に、2 重帰納操作を繰り返した 2 重帰納関

数です。一方、前節で示した様に、適切に n 変数アッカーマン関数を定義する事で、 n 重帰納関数を作ることができます。ここで、 n 重帰納関数はいかなる $n-1$ 重帰納関数よりも大きい、という明確な強弱関係を持ちます。

チェーン表記のように、一見、多重帰納のように見えてそうでない関数には、たとえばこのような関数があります。

$$A(0, 0, z) = z + 1 \quad (5.5)$$

$$A(0, y + 1, 0) = A(0, y, 1) \quad (5.6)$$

$$A(0, y + 1, z + 1) = A(0, y, A(x, y + 1, z)) \quad (5.7)$$

$$A(x + 1, y + 1, z + 1) = A(x, A(x + 1, y, z + 1), A(x + 1, y, z + 1)) \quad (5.8)$$

この式は、このように計算されます。

$$A(x + 1, y + 1, z + 1) = A(x, A(x + 1, y, z + 1), A(x + 1, y, z + 1)) \quad (5.9)$$

$$= A(x - 1, A(x, A(x + 1, y, z + 1) - 1, A(x + 1, y, z + 1)), A(x, A(x + 1, y, z + 1) - 1, A(x + 1, y, z + 1))) \quad (5.10)$$

$$= \dots \quad (5.11)$$

$$= A(0, A_2, A_2) \quad (A_2 \text{ は } x, y, z \text{ の } 2 \text{ 重帰納関数}) \quad (5.12)$$

$$= A(A_2, A_2) \quad (5.13)$$

このように、 $A(x, y, z)$ の計算は 2 重帰納です。3 項漸化式において、このように最初に x を減らして、次に y を減らすという計算方法では、2 重帰納どまりです。一方、3 変数アッカーマン関数では、最初に y を減らして、 y が 0 に到達してはじめて x を減らすことができます。そのために、 x を 1 減らす計算が y を減らす計算を数え上げることができます。この計算順序の違いが、2 重帰納か 3 重帰納かの差を生んでいます。このことが、多重帰納関数の計算を理解する上では重要です。

5.3 矢印回転表記とバード数

5.3.1 矢印回転表記

バード氏が考えたバードの矢印回転表記 (Bird's revolving arrow notation) とバード数 (Bird's number) について、巨大数探索スレッドでは色々と議論がされてきました。このバード氏のホームページは、uglypc.ggh.org.uk 上で公開されていましたが、すでに消えています。「巨大数探索スレッド 5」には、バードの矢印回転表記について、以下のように記録されています。

【定義】バードの矢印回転表記

7 名前 :【Bird's Revolving Arrow Notation】 : 03/04/04 19:46

定義をまとめておきます。

参考 : チェーンとの対応は $(a \ b \ \dots \ x \ y \ z) = {}_1(a, b, \dots, x, y, z)$

タワーとの対応は $(a \ \dots c \ \text{個} \ \dots \ b) = (a \ b \ c) = {}_1(a, b, c)$

以下 a, b, \dots, z は全て自然数 (> 0) とします。

まず多変数関数 ${}_1$ を次で定めます。

$${}_1(a) := a$$

$${}_1(a, b) := a^b$$

3 変数以上に対しては、

$${}_1(a, b, \dots, x, y, z) := {}_1(a, b, \dots, x, y)(y = 1 \text{ or } z = 1)$$

$${}_1(a, b, \dots, x, y, z) := {}_1(a, b, \dots, x, {}_1(a, b, \dots, x, y - 1, z), z - 1)(y > 1, z > 1)$$

次に多変数関数 ${}_{n-1}$ から、多変数関数 ${}_n$ を作ります ($n > 1$)。方法は、

$${}_n(a) := a$$

$${}_n(a, b) := a^b$$

$${}_n(a, b, c) := a^b(b = 1 \text{ or } c = 1)$$

$${}_n(a, b, 2) := {}_{n-1}(a, a, \dots, a)(a \text{ が } b \text{ 個})$$

$${}_n(a, b, c) := {}_n(a, {}_n(a, b - 1, c), c - 1)(b > 1, c > 2)$$

4 変数以上に対しては、

$${}_n(a, b, \dots, x, y, z) := {}_n(a, b, \dots, x, y)(y = 1 \text{ or } z = 1)$$

$${}_n(a, b, \dots, x, y, z) := {}_n(a, b, \dots, x, {}_n(a, b, \dots, x, y - 1, z), z - 1)(y > 1, z > 1)$$

ここで、 ${}_n(a, b, 2)$ の式を間違いやすいので、注意が必要です。そこだけ規則性が崩れています。

なお、この定義はオリジナルの表記を分かりやすい表記に書き直しています。オリジナルの表記と、上記定義との対応を示します。

$$a \ b \ \dots \ x \ y \ z = {}_1(a, b, \dots, x, y, z)$$

$$a \ b \ \dots \ x \ y \ z = {}_2(a, b, \dots, x, y, z)$$

$$a \ b \ \dots \ x \ y \ z = {}_3(a, b, \dots, x, y, z)$$

このように、タワー表記の、チェーン表記のを、さらに回転させていくことから矢印回転表記と呼ばれます。この後、さらにチェーンを回転させると、タワー表記と区別できなくなるため、1 回転した時点で (1) のように表記して、

$$a(-1)b(-1)\dots(-1)x(-1)y(-1)z = {}_4(a, b, \dots, x, y, z)$$

$$a(-n)b(-n)\dots(-n)x(-n)y(-n)z = {}_{4n}(a, b, \dots, x, y, z)$$

となります。さらに、タワー表記のように回転矢印を重ねることもできます。

$$a(n)[c]b = a(n)(n) \dots c \text{ 個} \dots (n)b = a(n)[c-1][a(n)[c]b-1]$$

$c > 2$ の定義は $[n+1](a,b,c)$ に吸収されるため、上記定義の中では $c = 2$ の定義式のみが記されています。

5.3.2 バード数

バード数は矢印回転表記をもとに、Bird 氏が作った巨大数ですが、これもオリジナルのホームページが消えています。巨大数探索スレッドには、以下の様に記録されています。

まずは $N = 3(G)[4]3$ (G はグラハム数) というものを考えます。

(G) とは G 回回転したチェーンのことで、

これはタワーと同じように演算子として用います。(チェーンの回転については省略)

$(G)[4] = (G)(G)(G)(G)$ です。

すなわち $N = 3(G)(G)(G)(G)3$ です。この N を下敷きにして、

$X(1) = N(N)[N]N$ を考えます。そしてこれから

$X(N) = X(N-1)(X(N-1))[X(N-1)]X(N-1)$ とパワーアップさせて、

この $X(N)$ を $X_1(N)$ と呼び直します。(ここ不正確かも)

そこで、

$$X_2(1) = X_1(N)$$

$$X_2(2) = (X_1)^2(N) = X_1(X_1(N))$$

$X_3(1) = X_2(N) = X_1^N(N)$... とビシビシ強化して、(この辺の詳細も省略)

$H = X_N(N)$ を考えます。ここからまた $X_H(N), (X(X_H(N)))(N), \dots$ のように X の添字部分に入れ子を作っていきます。

入れ子操作を $X_H(N)$ 回行った結果生まれる巨大数が「バード数」です。多分。

非常に複雑な定義ですが、 $X_m(n)$ を $X[m](n)$ と書き直すことで、この定義は以下の様に書く事ができます。

【定義】バード数

$$X[0](n) = n(n, n, n)$$

$$X[m+1](n) = X[m]^n(n)$$

$$N = G(3, 3, 4)$$

$$f(n) = X[n](N)$$

としたとき、バード数 $B = f^{f^2(N)+1}(N)$

ここで、 $H = f(n), X_H(N) = f^2(N)$ と計算されます。また、入れ子操作 n 回は $f^{n+1}(N)$ です。

5.3.3 バード数の大きさ

バードの回転矢印表記は、チェーン表記を元としているため、回転矢印の長さを1つ延ばす操作、すなわち変数を1つ増やす操作が、2重帰納操作です。したがって、変数の長さを数え上げる操作は3重帰納操作です。 $n(a, b, 2)$ の定義式は、 $n-1$ の変数の長さを数え上げているので、 $n-1$ から n を作る操作は3重帰納です。したがって、 n の n を変数として矢印の回転数を数え上げると、4重帰納関数ができます。

バード数の定義は、 $X[m](n)$ の定義において、4重帰納の矢印回転関数を n 回繰り返す操作を m 回繰り返しているため、4重帰納に2重帰納を加えています。さらに、バード数の生成では合成が加わっています。4重帰納、2重帰納、原始帰納、合成を組み合わせた4重帰納です。

チェーン、バードの矢印回転関数、ふいつしゅ数の比較については、巨大数探索スレッドで白熱した議論がされていますが、以上から、

1. 2重帰納: チェーンを伸ばす操作、 $F_1(x)$
2. 3重帰納: 矢印を回転させる操作、 $F_2(x)$
3. 4重帰納: 矢印回転関数、バード数

となります。

5.3.4 バード数とアッカーマンの比較

$g(n) = Xn$ とすると、
 $g(2) > N$ となって、
 $g^{n+1}(2) > f^n(N)$
 となります。したがって、
 $B = f^{f^2(N)+1}(N) < g^{g^3(2)+2}(2)$
 となります。ここで、
 $X[m](n) < A(1, 0, 0, m, n)$
 であることが、

$$X[0](n) = n(n, n, n)$$

$$X[m+1](n) = X[m]^n(n)$$

この漸化式と、

$$A(1, 0, 0, 0, n) = A(n, 0, 0, n) \quad n(n, n, n) = X[0]n$$

$$A(1, 0, 0, m+1, n) = A[m]^n(A(1, 0, 0, m+1, 0))(A[m](x) = A(1, 0, 0, m, x))$$

この漸化式から分かります。したがって、
 $g(n) = Xn < A(1, 0, 0, n, n) < A(1, 0, 1, 0, 2n)$
 となります。この式は、すなわち関数 $g(n) = Xn$ が、4重帰納1回(矢印回転関数)と2重帰納1回(n 重の入れ子操作)から計算されることから、導かれた式です。

バード数は $g(n)$ から原始帰納的に導かれるため、計算を単純化することで以下のように $A(1, 0, 1, 1, *)$ の形にすることができます。

$g^n(2) < A(1, 0, 1, 1, 2n)$ より、

$$\begin{aligned} B &= g^{g^3(2)+2}(2) < A(1, 0, 1, 1, A(1, 0, 1, 1, 6) * 2 + 4) \\ &< A(1, 0, 1, 1, A(1, 0, 1, 1, A(1, 0, 1, 2, 0))) \\ &= A(1, 0, 1, 2, 2) \end{aligned}$$

すなわち、 $B < A(1, 0, 1, 2, 2)$ となります。

したがって、たとえば $A(1, 1, 1, 1, 1)$ は、バード数よりもはるかに大きくなります。

$$\begin{aligned} A(1, 1, 1, 1, 1) &= A(1, 1, 1, 0, A(1, 1, 1, 1, 0)) \\ &= A(1, 1, 1, 0, A(1, 1, 0, 1, 1)) \\ &= A(1, 1, 1, 0, A(1, 1, 0, 0, A(1, 0, 0, 1, 1))) \\ &= A(1, 1, 1, 0, A(1, 1, 0, 0, A(1, 0, 0, 0, 3))) \\ &= A(1, 1, 1, 0, A(1, 1, 0, 0, A(2, 2, 3, 3))) \\ &= A(1, 1, 1, 0, A(1, 0, A(2, 2, 3, 3), 0, A(2, 2, 3, 3))) \\ &> A(1, 1, 1, 0, A(1, 0, 2, 1, 2)) \\ &> A(1, 1, 1, 0, B) \end{aligned}$$

5.4 ふいつしゅ数バージョン 3

ふいつしゅ数バージョン 3 (F_3) は、以下のように定義されます。

【定義】ふいつしゅ数バージョン 3

[1] 関数 $f(x)$ から $g(x)$ への写像 $s(n)$ ($n > 0$) を以下のように定める。

$$s(1)f := g; g(x) = f^x(x) \text{ [原始帰納]}$$

$$s(n)f := g; g(x) = [s(n-1)^x]f(x) (n > 1) \text{ [n 重帰納]}$$

[2] 関数 $f(x)$ から $g(x)$ への写像 $ss(n)$ ($n > 0$) を以下のように定める。

$$ss(1)f := g; g(x) = s(x)f(x) \text{ [帰納回数の数え上げ]}$$

$$ss(n)f := g; g(x) = [ss(n-1)^x]f(x) (n > 1) \text{ [さらに数え上げ]}$$

[3] ふいつしゅ関数 $F_3(x)$ を以下のように定める。

$$F_3(x) := ss(2)^{63}f; f(x) = x + 1$$

[4] ふいつしゅ数 $F_3 := F_3^{63}(3)$ とする。

この定義は、 F_1 や F_2 と比べると異なった表現が用いられていますが、内容は F_2 の拡張です。また、「巨大数探索スレッド」で定義された F_3 (旧 F_3) の定義とも、異なっています。旧 F_3 の定義の中で、最も本質的な部分を記述したものが新 F_3 です。旧 F_3 の定義については、煩雑になるためここには記しません。

そこで、 F_1, F_2 , 旧 F_3 , 新 F_3 の定義の変遷についてまとめます。

表 5.1: F_1 から F_3 までの定義の変遷

バージョン	定義
F_1	2重帰納操作である S 変換 (数と関数のペアから数と関数のペアへの写像) S 変換を多数回繰り返す操作 (SS 変換)
F_2	S 変換 (2重帰納) の回数を数え上げる SS 変換 (3重帰納) (SS 変換は数、関数、S 変換から数、関数、S 変換への写像)
旧 F_3	F_2 の S 変換, SS 変換, SSS... 変換の定義に相当する $s(n)$ 変換 (多重帰納) 帰納回数を数え上げる $ss(1)$ 変換
新 F_3	F_3 の $s(n)$ 変換の定義を、関数から関数への写像 (汎関数) へと簡略化 $s(1)$ 変換の定義 (F_1, F_2 の S 変換に相当) を、2重帰納から原始帰納へと簡略化

つまり、 F_1 から旧 F_3 まで、定義を拡張することでより高い帰納程度を持つ関数、そして巨大数を作ってきましたが、その結果、旧 F_3 の定義は非常に複雑なものになりました。新 F_3 では、旧 F_3 の定義の中から、高い帰納程度を持つ関数を作るために必要な定義のみを残して簡略化しました。旧 F_3 と新 F_3 の大きさは厳密には異なりますが、関数の帰納程度が同程度になります。

ここで、新 F_3 の定義の 2 つの簡略化について説明します。まずは、1 つ目の簡略化についてです。SS 変換は「数と関数と S 変換」から「数と関数と S 変換」への写像であり、S 変換の回数を数え上げることで、大きな関数を作っています。S 変換そのものは、S 変換の回数を増やすことで大きくしていますが、そのこと自体は関数の大きさを大きくする事に本質的に役立っていません。したがって、SS 変換の定義の中で、S 変換を大きくするしくみは「無駄なしくみ」ということになります。また、数を関数に代入する定義についても、関数さえ定義されれば最終的にできた関数に数を代入すればいいのですから、「無駄な定義」ということになります。このように、SS 変換の定義は「関数から関数への写像」つまり汎関数の定義の部分が本質的ということになります。そこで、 $s(n)$ 変換は汎関数を定義としました。

次に、新 F_3 の 2 つ目の簡略化について説明します。これまで、S 変換はアッカーマンを元にした 2 重帰納操作で定義されていました。そして、2 重帰納操作を数え上げることで 3 重帰納操作を、それをまた数え上げることで 4 重帰納操作を、という定義を重ねる事で、多重帰納操作が定義されました。ここで、2 重帰納操作そのものは、原始帰納操作の数え上げで定義できますから、元となる操作は 2 重帰納である必要はなく、原始帰納で十分ということになります。そうすれば、原始帰納の数え上げから 2 重帰納、その数え上げから 3 重帰納と、多重帰納を定義できます。そこで、新 F_3 の $s(1)$ は原始帰納定義としました。新 F_3 の $s(2)$ 、すなわち

$$s(1)f := g; g(x) = f^x(x)$$

$$s(2)f := g; g(x) = [s(1)^x]f(x)$$

から計算される $s(2)$ は、 F_1 や F_2 の S 変換とは異なりますが、同じ程度の大きさの 2 重帰納操作となります。 $s(1)f := g; g(x) = f^{x+1}(1)$ とすれば、 $s(2)$ は F_1 や F_2 の S 変換と一致します。

F_3 の $s(n)$ と多変数アッカーマン関数の大きさは、以下の様に比較できます。

$$f(x) = x + 1 \text{ とすると}$$

$$[s(1)^{a_1}][s(2)^{a_2}] \dots [s(n)^{a_n}]f(x) \approx A(a_n, \dots, a_2, a_1, x)$$

ここで、 $s(1)$ の定義として $s(1)f := g; g(x) = f^{x+1}(1)$ を採用すると両辺がイコールになります。このことは、 F_3 の $s(n)$ と多変数アッカーマン関数の漸化式を比較すると両者が一致することで示す事ができます。

つまり、多重帰納の計算手順を定義したものが多変数アッカーマン関数、多重帰納における「操作の数え上げ」を汎関数で表現したものが F_3 の $s(n)$ であり、両者は等価である、ということになります。

$ss(1)$ は、 $g(x) = s(x)f(x)$ によって関数 $f(x)$ に対する x 重帰納を定義しています。これは多重帰納の回数を数え上げるもので、 $f(x)=x+1$ としてこのように計算されます。

$$ss(1)f(1) = s(1)f(1) \approx A(1, 1)$$

$$ss(1)f(2) = s(2)f(2) \approx A(1, 0, 2)$$

$$ss(1)f(3) = s(3)f(3) \approx A(1, 0, 0, 3)$$

$$ss(1)f(4) = s(4)f(4) \approx A(1, 0, 0, 0, 4)$$

$$ss(1)f(5) = s(5)f(5) \approx A(1, 0, 0, 0, 0, 5)$$

...

$$ss(1)f(n) = s(n)f(n) \approx A(1, (0 \text{ が } n-1 \text{ 個}), n)$$

$ss(1)$ によって生成される x 重帰納関数 $ss(1)f(x)=s(x)f(x)$ は、どのような多重帰納関数よりも大きな関数です。ある i 重帰納関数 $g_i(x)$ を考えた時に、 $x > i$ において常に $s(x)f(x) > g_i(x)$ となるためです。つまり、 $ss(1)f(x)$ は多重帰納以上の関数であるということです。

$F_3(x)$ は、この $ss(1)$ という操作の数え上げ操作を 63 回繰り返したものです。このように、 $F_3(x)$ は多重帰納関数よりも大きな関数です。

5.5 巨大数の比較

本書は「巨大関数論」ではなくて「巨大数論」としてはいますが、ここまでは帰納関数の定義について議論が集中していました。そこで、これまでに登場した巨大数の大きさを比較します。Table 5.2 は、巨大数を一覧にしています。巨大数生成に使われる関数がどの程度の帰納関数であるかを「帰納の程度」として表しています。

表 5.2: 巨大数の大小比較 (下の数ほど大きい)

巨大数	帰納の程度
無量大数 = 10^{68}	原始帰納
エディントン数 = 136×2^{256}	原始帰納
不可説不可説転 = $10^7 \times 2^{122}$	原始帰納
グーゴルプレックス = $10^{10^{100}}$	原始帰納
3 3 3	2 重帰納
G: グラハム数	2 重帰納
3 3 3 3	2 重帰納
A(2,2,2)	2 重帰納
3 ... (G 個) ... 3	2 重帰納
F_1 : ふいつしゅ数 V1	2 重帰納
A(1,1,1,1)	3 重帰納
$3(3,3,3) = 3 \quad 3 \quad 3$	3 重帰納
F_2 : ふいつしゅ数 V2	3 重帰納
G(3,3,3)	3 重帰納
B: バード数	4 重帰納
A(1,1,1,1,1)	4 重帰納
A(1,...(B 個)...,1)	(B-1) 重帰納
F_3 : ふいつしゅ数 V3	多重帰納以上

第6章 順序数とHardy関数

6.1 多重帰納を超えて

全章で示したように、 $F_3(x)$ はいかなる多重帰納関数よりも大きな関数です。ここから先は、 $F_3(x)$ よりも大きな関数、そして巨大数を探求する事になります。では、そのように大きな関数を、どうやって比較するのでしょうか。

本書に登場する大きな関数は、計算によって値を求めることで大きさを比較することは現実的に不可能です。グラハム数ですら実際に計算することは無理ですから、それよりも大きな巨大数については関数の性質を調べて大きさを比較するよりありません。

多重帰納の範囲にある関数であれば、たとえば、3重帰納関数の $F_2(x)$ よりも、4重帰納関数の矢印回転関数の方が大きい、というように、それが何重帰納であるかを調べる事で、関数の大きさを比較できます。

では、多重帰納よりも大きな関数を定義したときに、その大きさをどのように比較するのでしょうか。

そのための有効な手段が、本章で解説する Hardy 関数です。Hardy 関数を理解するためには、まず「順序数」の理解が必要です。Hardy 関数は、関数の大きさを順序数によって階層づけることができます。多重関数以上の関数についても、その大きさを相当する Hardy 関数によって対応づけられる順序数の大きさによって比較する事が可能となります。

そこで、本章ではまず順序数と Hardy 関数について解説し、これまでに出てきた関数の大きさを Hardy 関数を用いて評価します。次章以降では、様々な関数の大きさを主として Hardy 関数 $H[\](x)$ または $F[\](x)$ を用いて評価します。

6.2 順序数

Wikipedia には、順序数について以下のように定義されています。

【定義】順序数

順序数（じゅんじょすう）とは、順序構造すなわち集合の元（要素、element）の間の序列のつけ方をあらかず順序型の特別なもののことである。

n 個の元からなる有限集合の順序数は、自然数の集合 $1, 2, \dots, n$ に通常の大小関係で序列をつけたものであり、これを集合の元の数と同じ文字 n で表す。このように自然数の n と順序数の n はまったく別のものであるが、（後に定める順序数の演算とともに）これを同一視して順序数は自然数の拡張（の一つ）であると見なす。

順序数には、有限順序数と無限順序数（超限順序数）の2種類があります。ここでは、自然数は有限順序数であると考えます。自然数以外の順序数は無限順序数です。無限順序数の中で最小のものを ω と呼びます。そうすると、数列 $1, 2, 3, 4, 5, \dots$ は ω に収束します。このとき、 $1, 2, 3, 4, 5, \dots$ を ω に収束する収束列と呼びます。 ω に対する収束列は、1種類とは限りません。たとえば、数列 $2, 4, 6, \dots$ も ω への収束列となります。

最小の無限順序数 ω に対して、 $\omega + 1$, $\omega + 2, \dots$ といった順序数を定義することができます。ここで、 α が順序数のとき、ある順序数 β が存在して $\beta = \alpha + 1$ となるならば、 β は後続順序数 (successor ordinal) であると言います。後続順序数でない順序数は極限順序数 (limit ordinal) と呼びます。 ω は極限順序数で、 $\omega + a$ ($a=1, 2, 3, \dots$) は後続順序数です。極限順序数は、基本列によって定義されます。

$\omega + 1$, $\omega + 2, \dots$ という基本列を考えると、この数列は ω に収束します。したがって、 ω は極限順序数です。この時に、 ω のことを $\omega \times 2$ と書きます。ここで、順序数に関しては通常の交換法則が成り立たず、 $2 \times \omega = \omega$ となりますので、 2 ではなく $\omega \times 2$ と表記する必要があります。

極限順序数 $\omega \times 2$ に対して、後続順序数 $\omega \times 2 + a$ ($a=1, 2, 3, \dots$) を定義できます。そして、 $\omega \times 3$ は $\omega \times 2 + 1$, $\omega \times 2 + 2, \dots$ を基本列とする極限順序数です。このように、後続順序数の定義 ($+1$) と極限順序数の定義を繰り返す事で、順序数が定義されます。基本列 a_1, a_2, a_3, \dots の極限順序数 ω を $\omega = \lim(a_1, a_2, a_3, \dots)$ と表記することになると、以下のように順序数が定義されます。

$$\begin{aligned} \omega \times 3 &= \lim(\omega \times 2, \omega \times 2 + 1, \omega \times 2 + 2, \dots) \\ \omega \times 4 &= \lim(\omega \times 3, \omega \times 3 + 1, \omega \times 3 + 2, \dots) \\ \omega \times a &= \lim(\omega \times (a-1), \omega \times (a-1) + 1, \omega \times (a-1) + 2, \dots) \\ \omega^2 &:= \omega \times \omega = \lim(\omega \times 2, \omega \times 3, \dots) \\ \omega^2 + \omega &= \lim(\omega^2, \omega^2 + 1, \omega^2 + 2, \dots) \\ \omega^2 + \omega \times a &= \lim(\omega^2 + \omega \times (a-1), \omega^2 + \omega \times (a-1) + 1, \omega^2 + \omega \times (a-1) + 2, \dots) \\ \omega^2 \times 2 &= \omega^2 + \omega^2 = \lim(\omega^2, \omega^2 + \omega, \omega^2 + \omega \times 2, \dots) \\ \omega^3 &= \omega^2 \times \omega = \lim(\omega^2, \omega^2 \times 2, \omega^2 \times 3, \dots) \\ &= \lim(\omega^2, \omega^3, \omega^4, \dots) \\ \omega \times 2 &= \omega + \omega = \lim(\omega, \omega + 2, \omega + 3, \dots) \\ \omega^{+1} &= \omega \times \omega = \lim(\omega \times 2, \omega \times 3, \dots) \\ \omega^{+2} &= \omega^{+1} \times \omega = \lim(\omega^{+1}, \omega^{+1} \times 2, \omega^{+1} \times 3, \dots) \\ \omega^2 &= \omega + \omega = \lim(\omega, \omega + 1, \omega + 2, \dots) \\ \omega^2 &= \omega \times \omega = \lim(\omega, \omega \times 2, \omega \times 3, \dots) \\ &= \lim(\omega, \omega^2, \omega^3, \dots) \end{aligned}$$

ここで、 $\omega^a := \omega \uparrow^a \omega$ (a 個の ω) と表記することにして、極限順序数 ω_0 は $\omega_0 = \lim(\omega, \omega^2, \omega^3, \omega^4, \omega^5, \dots)$ と定義されます。 ω_0 は、 ω から有限回の加算・乗算・冪状では到達できない最小の無限順序数です。

6.3 Hardy 関数

Hardy 関数は、以下の様に定義されます。

【定義】 Hardy 関数

$$\begin{aligned} H[0](x) &= x \\ H[\alpha + 1](x) &= H[\alpha](x + 1) \\ H[\alpha](x) &= H[\alpha_x](x) \quad (\alpha \text{ が極限順序数の時}) \end{aligned}$$

順序数の定義は、後続順序数の定義 (順序数に+1 する) と、極限順序数の定義 (順序数の極限を取る) の組み合わせです。その定義を、それぞれ関数に $x+1$ を代入する操作と、対角化の操作に対応させることにより、関数の階層性を順序数の階層性と対応づけています。

ここで、定義の中に基本列が用いられていますが、基本列の取り方は一様ではないため、 の定義にあらわれるすべての基本列の取り方を厳密に定めて、はじめて $H[\]$ が定義されたこととなります。ただし、基本列のとり方による増大度の違いよりは、順序数の大きさによる増大度の方がはるかに大きい為、基本列の取り方いかんに関わらず、 $\alpha > \beta$ であれば $H[\alpha] > H[\beta]$ が成り立つため、 $H[\text{順序数}]$ と記述して大体の大きさを表します。

また、基本列の中でなんらかの方法で 1 つの正規な基本列 (canonical fundamental sequence) を与え、 $H[\alpha_x]$ の正規な基本列であるとする事で、 $H[\]$ の定義は一意に定まります。

$f(x) = H[\alpha](x)$ としたときに ($f(x) = x + 1$ のときに $\alpha = 1$)

$H[\alpha \times 2](x) = f(f(x))$ (合成)

$H[\alpha \times n](x) = f^n(x)$ (合成 n 回)

$H[\alpha \times \omega](x) = f^x(x)$ (原始帰納)

$H[\alpha \times \omega^2](x) = f^{f^x(x)}(f^x(x))$ (原始帰納 2 回)

$H[\alpha \times \omega^\omega](x)$: f の 2 重帰納関数

$H[\alpha \times \omega^{\omega \times 2}](x)$: 2 重帰納 2 回

$H[\alpha \times \omega^{\omega \times n}](x)$: 2 重帰納 n 回

$H[\alpha \times \omega^{\omega^2}](x)$: 3 重帰納

$H[\alpha \times \omega^{\omega^2 \times n}](x)$: 3 重帰納 n 回

$H[\alpha \times \omega^{\omega^3}](x)$: 4 重帰納

$H[\alpha \times \omega^{\omega^n}](x)$: $n + 1$ 重帰納

$H[\alpha \times \omega^{\omega^\omega}](x)$: x 重帰納 (多重帰納以上)

また、以下のような $F[\](x)$ を定義することにより、

【定義】Hardy 関数の別表記

$F[0](x) = x + 1$

$F[\alpha + 1](x) = F[\alpha]^x(x)$ (α が後続順序数の時)

$F[\alpha](x) = F[\alpha_x](x)$ (α が極限順序数の時)

$f(x) = F[\alpha](x)$ として ($f(x) = x + 1$ のときに $\alpha = 0$)

$F[\alpha + 1](x) = f^x(x)$ (原始帰納)

$F[\alpha + \omega](x)$: 2 重帰納

$F[\alpha + \omega^2](x)$: 3 重帰納

$F[\alpha + \omega^n](x)$: $n + 1$ 重帰納

$F[\alpha + \omega^\omega](x)$: x 重帰納 (多重帰納以上)

といったように、

$F[\](x) \quad H[\](x)$

となり、表記が若干楽になります。巨大数探索スレッドでは、主としてこの表記が使われています。

6.4 色々な関数の Hardy による近似

F_3 の計算: $f(x) = F[\alpha](x)$ として

$$ss(1)f(x) = F[\alpha + 1](x)$$

$$ss(1)^2 f(x) = F[\alpha + 2](x)$$

$$ss(2)f(x) = ss(1)^x f(x) = F[\alpha + 2 \times x](x) = F[\alpha + x + 1](x)$$

$$F_3(x) = ss(2)^{63} f(x) = F[\alpha + 63 \times 63](x)$$

なお、ここから先は

$$ss(3)f(x) = ss(2)^x f(x) = F[\alpha + x + 2](x)$$

$$ss(4)f(x) = ss(3)^x f(x) = F[\alpha + x + 3](x)$$

$$ss(x)f(x) = F[\alpha + x^2](x)$$

このように計算されます。

表 6.1: 色々な関数の Hardy による近似 (N は大きな整数)

関数	$H[\]$ による近似	$F[\]$ による近似
<i>Knuth</i> の上矢印 $3(n)3$	$H[\](n)$	$F[\](n)$
$A(n, 3)$	$H[\](n)$	$F[\](n)$
F_1, F_2 の S 変換 1 回	$H[\](n)$	$F[\](n)$
$F_1(n)$	$H[\times N](n)$	$F[\times N](n)$
n 個の n からなる Conway のチェーン	$H[{}^2](n)$	$F[{}^2](n)$
F_3 の $s(2)n$ 回 F_1, F_2 の S 変換 n 回	$H[{}^2](n)$	$F[{}^2](n)$
$F_2(n)$	$H[{}^{2 \times 63}](n)$	$F[{}^{2 \times 63}](n)$
チェーンの回転 $n(3, 3, 3)$	$H[{}^3](n)$	$F[{}^3](n)$
F_3 の $s(3)n$ 回 F_2 の SS 変換 n 回	$H[{}^3](n)$	$F[{}^3](n)$
バード数の $X(n, 1, 1)$	$H[{}^{3+1}](n)$	$F[{}^{3+1}](n)$
$A(1, 0, 2, 1, n)$	$H[{}^{3+\times 2+1}](n)$	$F[{}^{3+\times 2+1}](n)$
$A(1, 0, 0, 0, 0, n) = A(n, 0, 0, 0, n)$	$H[{}^4](n)$	$F[{}^4](n)$
$A(n, n, n, n, n)$	$H[{}^4](n)$	$F[{}^4](n)$
F_3 の $s(4)n$ 回	$H[{}^4](n)$	$F[{}^4](n)$
$A(n$ 個の n) or $A(n$ 個の 1)	$H[\](n)$	$F[\](n)$
F_3 の $s(n)$	$H[\](n)$	$F[\](n)$
$F_3(n)$	$H[{}^{+1 \times 63}](n)$	$F[{}^{+1 \times 63}](n)$

第7章 証明可能帰納関数

7.1 証明可能帰納関数

証明可能帰納関数 (provably recursive function) とは、ペアノ算術 (Peano arithmetic) という体系の中で証明可能な関数のこと、と定義されています。 $n < 0$ の時に、Hardy 関数 $H[n]$ および $F[n]$ が証明可能帰納関数となることが証明されているようです。 $H[0]$ は証明可能帰納関数ではありません。本章では、 $H[0]$ 以下の関数について扱います。

7.2 2重リストアッカーマン関数

たろう氏は、多変数アッカーマン関数を2重リストに拡張して、2重リストアッカーマン関数を定義しました。多変数アッカーマンの n 個目の引数が a であることを、 $[n, a]$ と表現するようにし、 $[]$ の中を多変数化することで拡張しました。これは、 $F[n](a)$ 位の増大度の関数となります。

【定義】2重リストアッカーマン関数

n : 0 個以上の 0
 X : 0 個以上の 0 以上の整数
 Y, Y_1, Y_2 : 0 個以上の 0 以上の整数リスト
 a, b, c : 0 以上の整数
 N : 十分大きな整数に対し、
 $index[... , b_3, b_2, b_1, b_0, a_0] = ... + N^3 \cdot b_3 + N^2 \cdot b_2 + N \cdot b_1 + b_0$ とし、
 $Ak()$ の中のリストは $index$ の大きい順に表記する。
 また、 $index$ が同じとなるリストは $Ak()$ 内に複数含まないとする。
 $Ak(X_1, [X, 0], X_2) = Ak(Y_1, Y_2)$
 $Ak(X_1, [, X], X_2) = Ak(Y_1, [X], Y_2)$
 $Ak([a]) = a + 1$
 $Ak(Y, [1, b + 1]) = Ak(Y, [1, b], [1])$
 $Ak(Y, [1, b + 1], [a + 1]) = Ak(Y, [1, b], [Ak(Y, [1, b + 1], [a])])$
 $Ak(Y, [X, c + 1, b + 1], [a]) = Ak(Y, [X, c + 1, b], [X, c, a], [a]) \dots X \quad or \quad c = 0$
 $Ak(Y, [X, c + 1, 0, , b + 1], [a]) = Ak(Y, [X, c + 1, 0, , b], [X, c, a, , a], [a])$

以下のように計算されています。

$$\begin{aligned}
 Ak([n, 1], [n]) &= Fn \\
 Ak([1, 0, 1], [n]) &= Fn \\
 Ak([a, 0, 1], [n]) &= F[(a \cdot n)](n) \\
 Ak([n, 0, 1], [n]) &= F[n^2](n)
 \end{aligned}$$

$$\begin{aligned}
Ak([1, 0, 0, 1], [n]) &= F[\alpha + 2](n) \\
Ak([a, 0, 0, 1], [n]) &= F[\alpha + 2, a](n) \\
Ak([n, 0, 0, 1], [n]) &= F[\alpha + 3](n) \\
Ak([1, 0, 0, 0, 1], [n]) &= F[\alpha + 3](n) \\
Ak([1, 0, 0, 0, 0, 1], [n]) &= F[\alpha + 4](n) \\
Ak([1, 0, 0, 0, 0, 0, 1], [n]) &= F[\alpha + 5](n) \\
Ak(\dots, [3, a3], [2, a2], [1, a1], [0, a0]) &= Ak(a3, a2, a1, a0) \\
Ak(\dots, [b3, b2, b1, b0, a], [n]) &= F[\alpha + 3 \cdot b3 + 2 \cdot b2 + \dots + b1 + b0, a](n) \\
Ak([n \text{ 個の } 1], [n]) &= F[\alpha + n](n)
\end{aligned}$$

7.3 ふいつしゅ数バージョン3の拡張

前章で、 F_3 の定義において

$$ss(x)f(x) = F[\alpha + \times 2](x)$$

と計算がされました。ここで、 $s(x)$ を3変数化して

$$s(1, 1, 1) := s(1)$$

$$s(a, 1, 1)f := g; g(x) = s(a - 1, x, x)f(x) (a > 1)$$

$$s(a, b, 1)f := g; g(x) = s(a, b - 1, x)f(x) (b > 1)$$

$$s(a, b, c)f := g; g(x) = [s(a, b, c - 1)]^x f(x) (c > 1)$$

とします。すなわち、 $s(1, 1, n) = s(n)$, $s(1, 2, n) = ss(n)$ です。このように定義すると、以下のような計算ができます。

$$s(1, 2, x)f(x) = ss(x)f(x) = F[\alpha + \times 2](x)$$

$$s(1, 3, 1)f(x) = s(1, 2, x)f(x) = F[\alpha + \times 2](x)$$

$$s(1, 3, 2)f(x) = s(1, 3, 1)^x f(x) = F[\alpha + \times 2 + 1](x)$$

$$s(1, 3, 3)f(x) = s(1, 3, 1)^{x^2} f(x) = F[\alpha + \times 2 + 2](x)$$

$$s(1, 3, x)f(x) = F[\alpha + \times 3](x)$$

$$s(1, x, x)f(x) = F[\alpha + \times 2](x)$$

$$s(2, 1, 1)f(x) = s(1, x, x)f(x) = F[\alpha + \times 2](x)$$

$$s(2, 1, 2)f(x) = F[\alpha + \times 2 + 1](x)$$

$$s(2, 2, 1)f(x) = s(2, 1, x)^x f(x) = F[\alpha + \times 2 + 1](x)$$

$$s(2, 2, x)f(x) = F[\alpha + \times 2 + \times 2](x)$$

$$s(2, 3, x)f(x) = F[\alpha + \times 2 + \times 3](x)$$

$$s(3, 1, 1)f(x) = s(2, x, x)f(x) = s(2, 2, x)^x f(x) = F[\alpha + \times 2 + \times 2](x)$$

$$s(x, 1, 1)f(x) = F[\alpha + \times 3](x)$$

このように、3変数の $s(x)$ 変換 $s(x, x, x)$ の大きさが $F[\alpha + \times 3](x)$ 程度なので、 x 変数の $s(x)$ 変換 $s(x, x, \dots, (x \text{ 個}), \dots, x)$ を考えると、その大きさは $F[\alpha + \times n](x)$ 程度となります。

7.4 多重リストアッカーマン関数

たろう氏は、多変数アッカーマンから2重リストアッカーマンに拡張したのと同様の拡張を繰り返すことで、多重リストアッカーマンを定義しました。その大きさは、以下のように計算されています。

【定義】多重リストアッカーマン関数

3重リスト

$$Ak([1, 0, 1], [1], [[n]]) = F[]_2(n)$$

$$Ak([1, 0, 0, 1], [1], [[n]]) = F[]_3(n)$$

$$Ak([1, 0, 0, 0, 1], [1], [[n]]) = F[]_4(n)$$

$$Ak([\dots, [3, a3], [2, a2], [1, a1], [0, a0]], [[n]]) = Ak([\dots, a3, a2, a1, a1], [n])$$

$$Ak([n \text{ 個の } 1], [1], [[n]]) = F[]_n(n)$$

4重リスト

$$Ak([1, 0, 1], [1], [[1]], [[n]]) = F[]^{4}_2(n)$$

$$Ak([1, 0, 0, 1], [1], [[1]], [[n]]) = F[]^{4}_3(n)$$

$$Ak([1, 0, 0, 0, 1], [1], [[1]], [[n]]) = F[]^{4}_4(n)$$

$$Ak([\dots, [3, a3], [2, a2], [1, a1], [0, a0]], [[n]]) = Ak([\dots, a3, a2, a1, a1], [n])$$

$$Ak([n \text{ 個の } 1], [1], [[1]], [[n]]) = F[]^{4}_n(n)$$

n重リスト

n重リストとすることで $F[]_0(n)$ に到達

7.5 ふいつしゅ数バージョン5

バージョン3の次は、バージョン4ではなくバージョン5の説明をします。巨大数探索スレッドでは、バージョン4が先に登場するのですが、この文章では、章を進めるにしたがってより大きな数、関数が登場するように記述するため、バージョン5よりも大きいバージョン4の説明は後回しにします。

「関数から関数への写像を考える」という考えをさらにすすめると、「関数から関数への写像」から「関数から関数への写像」への写像を考えることができます。では、このような写像に対して写像を適用する回数の数え上げを定義する事はできないでしょうか。このような考察を進めた結果、以下の様なふいつしゅ数バージョン5が定義されました。

[1] 集合 $M_n (n = 0, 1, 2, \dots)$ を以下のように定める。

$M_0 =$ 自然数の集合

$M_{n+1} =$ 写像 M_n M_n 全体の集合

M_n の元を M_n 変換と呼ぶ

[2] M_n 変換 $m(n) (n \geq 1)$ を以下のように定める。

$f_n \in M(n)$ に対して、 $m(n+1)(f_n) = g_n$ を以下で定める。

$f_{n-1} \in M(n-1)$ に対して、 $g_n(f_{n-1}) = g_{n-1}$ を以下で定める。

$f_{n-2} \in M(n-2)$ に対して、 $g_{n-1}(f_{n-2}) = g_{n-2}$ を以下で定める。

.....

$f_0 \in M(0)$ に対して、 $g_1(f_0) = g_0$ を以下で定める。

$$g_0 = (..((f_n^{f_0} f_{n-1}) f_{n-2}) \dots f_1) f_0$$

すなわち

$$m(1)(f_0) = f_0^{f_0} \tag{7.1}$$

$$(m(2)f_1)f_0 = (f_1^{f_0})(f_0) \tag{7.2}$$

$$(..((m(n+1)f_n)f_{n-1}) \dots f_1)f_0 := (..(f_n^{f_0} f_{n-1}) \dots f_1)f_0 \tag{7.3}$$

[3] ふいつしゅ関数 $F_5(x)$ を以下のように定める。

$$F_5(x) := ((..((m(x)m(x-1))m(x-2)) \dots m(2))m(1))(x) \tag{7.4}$$

[4] ふいつしゅ数 $F_5 := F_5^{63}(3)$ とする。

定義 [2] より、 $m(1)(x) := x^x$ です。また、

$$(m(2)f)(x) = (f^x)(x)$$

となりますから、 $m(2)$ は F_3 における $s(1)$ の定義と一致し、

$$m(2) = s(1)$$

となります。さらに、

$$((m(3)m(2))f)(x) = (m(2)^x f)(x) = (s(1)^x f)(x) = (s(2)f)(x)$$

すなわち、 $m(3)m(2) = s(2)$

$$((m(3)^2 m(2))f)(x) = ((m(3)(m(3)m(2)))f)(x) = ((m(3)m(2))^x f)(x) = (s(2)^x f)(x) = (s(3)f)(x)$$

すなわち、 $m(3)^2 m(2) = s(3)$

$$m(3)^n m(2) = s(n)$$

$$m(4)m(3)m(2) = ss(1)$$

といったように、計算ができます。すなわち、 $m(3)$ は操作の対角化で、 $m(2)=s(1)$ の原始帰納に対して、 $m(3)$ を繰り返し適用することで $m(3)^n m(2) = s(n)$ の多重帰納が生成されます。それを数え上げるのが $m(4)$ ということです。

Hardy 関数との対応を考えると、 $m(2)$ の定義から $f(x) = F[\](x)$ の時、 $m(2)f(x) = F[\ +1](x)$ となりますが、これを

$$m(2) = F[\ +1]$$

と便宜的に表記することとします。すると、

$$m(3)m(2) = s(2) = F[\ +\]$$

$$m(3)m(3)m(2) = s(3) = F[\ +\ ^2]$$

$$m(3)^a m(2) = s(a) = F[\ +\ ^a]$$

$$m(4)m(3)m(2) = ss(1) = s(x) = F[\ +\ \]$$

となります。さらに、

$$m(3)[m(4)m(3)]m(2) = ss(2) = F[\ +\ \ +1]$$

$$m(3)[m(3)m(4)m(3)]m(2) = ss(3) = F[\ +\ \ +2]$$

$$m(3)^a m(4)m(3)m(2) = ss(a+1) = F[\ +\ \ +a]$$

$$m(4)m(3)m(4)m(3)m(2) = ss(n) = F[\ +\ \ \times 2]$$

ここまでが、 F_3 の定義で記述できるところです。さらに、 F_3 の拡張である多変数化した $s(x)$ を使うことで、

$$[m(4)m(3)]^2 m(2) = ss(n) = s(1, 3, 1) = F[\ +\ \ \times 2]$$

$$[m(4)m(3)]^3 m(2) = s(1, 4, 1) = F[\ +\ \ \times 3]$$

$$[m(4)m(3)]^a m(2) = s(1, a+1, 1) = F[\ +\ \ \times a]$$

$$[m(4)m(4)m(3)]m(2) = [m(4)(m(4)m(3))]m(2) = s(2, 1, 1) = F[\ +\ \ \ ^2]$$

$$m(3)[m(4)m(4)m(3)]m(2) = s(2, 1, 2) = F[\ +\ \ \ ^2+1]$$

$$m(4)m(3)[m(4)m(4)m(3)]m(2) = s(2, 2, 1) = F[\ +\ \ \ ^2+]$$

$$m(3)m(4)m(3)[m(4)m(4)m(3)]m(2) = s(2, 3, 1) = F[\ +\ \ \ ^2+ \times 2]$$

$$m(3)^2 m(4)m(3)[m(4)m(4)m(3)]m(2) = s(2, 4, 1) = F[\ +\ \ \ ^2+ \times 3]$$

$$m(4)m(4)m(3)[m(4)m(4)m(3)]m(2) = s(3, 1, 1) = F[\ +\ \ \ ^2 \times 2](x)$$

$$[m(4)m(4)m(3)]^3 m(2) = s(4, 1, 1) = F[\ +\ \ \ ^2 \times 3](x)$$

$$m(4)^3 m(3)m(2) = m(4)[m(4)m(4)m(3)]m(2) = s(x, 1, 1) = F[\ +\ \ \ ^3]$$

$$m(4)^4 m(3)m(2) = s(x, 1, 1, 1) = F[\ +\ \ \ ^4]$$

$$m(5)m(4)m(3)m(2) = s(1, 1, \dots(1 \text{ が } x \text{ 個}), 1, 1) = F[\ +\ \ \]$$

このように計算されます。

これまでの計算から、

$$m(3)m(2) = F[\ +\ \]$$

$$m(4)m(3)m(2) = F[\ +\ \ \]$$

$$m(5)m(4)m(3)m(2) = F[\ +\ \ \ \]$$

となり、同様に

$$m(6)m(5)m(4)m(3)m(2) = F[\ +\ \ \ \ \ ^4]$$

$$m(7)m(6)m(5)m(4)m(3)m(2) = F[\ +\ \ \ \ \ \ ^5]$$

...

となることが期待されます。この時、

$$F_5(x) = F[\ \ 0]$$

となります。

7.6 ヒドラゲーム

たろう氏による Hardy 関数とヒドラゲームの関係に関する考察。

$$H'[0](n) = n$$

$$H'[a+1](n) = H'[a](n+1)$$

$$H'[A](n) = H'[A_{(n+1)}](n+1) \dots\dots (A \text{ は極限順序数、} A_n \text{ はその収束列})$$

収束列を、

$$\hat{a}^{(a+1)} = \lim \hat{a} \cdot n$$

$$\hat{A} = \lim \hat{A}_n \dots\dots (A \text{ は極限順序数、} A_n \text{ はその収束列})$$

$$b+A = \lim (b+A_n) \dots\dots (A \text{ は極限順序数、} A_n \text{ はその収束列})$$

と定義し、

ヒドラゲームの{ } 中の文字列を、

$$"[+ a +]" \Rightarrow \hat{a}$$

$$b + a \Rightarrow b + a$$

$$"" \Rightarrow 0$$

ヒドラゲームの{ } 内の文字列 \Rightarrow Hardy Function の順序数

文字列の+で文字列の結合を表す

のように順序数に変換すれば、

ヒドラゲームと、 $H'[\hat{x}, +, 0$ で作られる $_{-0}$ 未満の順序数](n) の値は完全に一致する。

8.3 ふいつしゆ数バージョン6

ふいつしゆ数バージョン6 (F_6) と同時に定義されるふいつしゆ関数バージョン6 は、 $H[0]$ 程度の大きさを持ちます。Hardy 関数、順序数やチューリングマシン (後の章で解説) の概念を使わずに作られた関数の中では、本書の中で、また本書最終更新日までの間に巨大数スレッドで検討された関数の中で、最も大きな関数です。それだけに、定義と計算方法は複雑かつ難解となっています。

ふいつしゆ数バージョン6 は、以下の様に定義されます。

【定義】ふいつしゆ数バージョン6

[1] 集合 $M[m, n]$ ($m = 0, 1, \dots; n = 1, 2, \dots$) を以下のように定める。

$M[0, 1]$ = 自然数から自然数への関数

$M[m + 1, 1] = M(m, n)$ ($n = 1, 2, \dots$) の元1個ずつを要素に持つ集合

$M[m, 1]$ の元は、その要素の要素の...要素である $M[0, 1]$ の元と同じ関数の働きを持つ。

$M[m, n + 1]$ = 写像 $M[m, n]$ $M[m, n]$ 全体の集合 ($n = 1, 2, \dots$)

[2] $M[m, n]$ の元 $m(m, n)$ を以下のように定める。ただし、 a_i, b_i, f_i は $m(m, i)$ の元とし、厳密な定義の構造は F_5 と同じである。

$m(0, 1)(x) := x + 1$

$m(m, n + 1)f_n f_{n-1} \dots f_1(x) := f_n^x f_{n-1} \dots f_1(x)$

($m = 0; n = 1, 2, \dots$ および $m = 1, 2, \dots; n = 2, 3, \dots$)

$m(m + 1, 1) := [m(m, 1), m(m, 2), m(m, 3), \dots]$

$m(m + 1, 2)[a_1, a_2, \dots] := [b_1, b_2, \dots]$ の b_n を以下で定める。

$b_n f_{n-1} \dots f_1(x) := a_y a_{y-1} \dots a_n f_{n-1} \dots f_1(x) (y = \max(x, n))$

[3] ふいつしゆ関数 $F_6(x)$ を以下のように定める。

$F_6(x) := m(x, 2)m(x, 1)(x)$

[4] ふいつしゆ数 $F_6 := F_6^{63}(3)$ とする。

F_6 の大きさは、以下の様に計算できます。

$m(1, 1) = [m_1, m_2, m_3, \dots] (m_i = m(0, i))$ すなわち $m(1, 1) = x + 1 = H[1]$

$m(1, 2)m(1, 1) = [a_1, a_2, a_3, \dots]$ とすると

$a_1(x) = m_x m_{x-1} \dots m_1(x)$ より、 F_5 と同様の計算で

$a_1 \approx H[0]$ すなわち $m(1, 2)m(1, 1) \approx H[0]$

$a_2 f_1(x) = m_y m_{y-1} \dots m_2 f_1(x) (y = \max(x, 2))$ より

$m_2 = H[\mathbf{x}] (H[a] \quad H[a \times]$ の変換)

$m_3 m_2 = H[\mathbf{x} \quad]$

$m_3 m_3 m_2 = H[\mathbf{x} \quad ^2]$

$m_4 m_3 m_2 = H[\mathbf{x} \quad]$

$m_5 m_4 m_3 m_2 = H[\mathbf{x} \quad]$

...

$a_2 = H[\mathbf{x} \quad 0]$

$a_3 f_2 f_1(x) = m_y m_{y-1} \dots m_3 f_2 f_1(x) (y = \max(x, 3))$ より、

$f_2 = H[\mathbf{x} \quad a]$ とすると

$m_3 f_2 = H[\mathbf{x} \quad a \quad]$

$m_3 m_3 m_2 = H[\mathbf{x} \quad a \quad ^2]$

$m_4 m_3 f_2 = H[\mathbf{x} \quad a \quad]$

$m_5 m_4 m_3 f_2 = H[\mathbf{x} \quad a \quad]$

...

$a_3 f_2 = H[\mathbf{x} \quad a \quad 0]$ となるため、

a_3 は $H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a \quad 0]$ の変換です。

$a_4 f_3 f_2 f_1(x) = m_y m_{y-1} \dots m_4 f_3 f_2 f_1(x) (y = \max(x, 4))$ より、 $f_2 = H[\mathbf{x} \quad a]$ 、 $f_3 = H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b]$ の変換とすると

$f_2 = H[\mathbf{x} \quad a]$

$f_3 f_2 = H[\mathbf{x} \quad a^b]$

$m_4 f_3 f_2 = H[\mathbf{x} \quad a^b \quad]$

$m_5 m_4 f_3 f_2 = H[\mathbf{x} \quad a^b \quad]$

...

$a_4 f_3 f_2 : H[\mathbf{x} \quad a^b \quad 0]$ となるため、

a_4 は $[H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b]] \quad [H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b \quad 0]]$ の変換です。このことから、

$a_4 a_3 : H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a \quad 0]$ の変換

$a_4 a_3 a_2 : H[\mathbf{x} \quad 0 \quad 0 \quad 0]$

と計算され、以下同様に

a_5 は $[H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b]] \quad [H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b]] \quad [H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b]] \quad [H[\mathbf{x} \quad a] \quad H[\mathbf{x} \quad a^b \quad 0]]$ の変換となつて、

$a_5 a_4 a_3 a_2 : H[\mathbf{x} \quad 0 \quad 0 \quad 0 \quad 0] = H[\mathbf{x} \quad 0 \quad \wedge \wedge 4]$

$a_6 a_5 a_4 a_3 a_2 : H[\mathbf{x} \quad 0 \quad \wedge \wedge 5]$

$a_{n+1} a_n \dots a_2 : H[\mathbf{x} \quad 0 \quad \wedge \wedge n]$

と計算され、

$$m(1,2)^2 m(1,1) \quad H[0^{^^}] = H[1]$$

となります ($1 = 0^{^^}$ の計算は前節)。

$m(1,2)^3 m(1,1) = [b_1, b_2, b_3, \dots]$ とすると、 b_i は上記 a_i の 0 を 1 に変えた式となります。したがって、

$$m(1,2)^3 m(1,1) \quad H[1^{^^}] = H[2]$$

$$m(1,2)^4 m(1,1) \quad H[2^{^^}] = H[3]$$

といったように、 $m(1,2)$ は $H[a]$ の a に 1 を足す効果を持ちます。このことから、 F_5 の計算と同様の構造で、

$$m(1,3)m(1,2)m(1,1)(x) \quad H[]$$

$$m(1,4)m(1,3)m(1,2)m(1,1)(x) \quad H[]$$

$$m(1,5)m(1,4)m(1,3)m(1,2)m(1,1)(x) \quad H[]$$

$$m(1,x)m(1,x-1)\dots m(1,2)m(1,1)(x) \quad H[0]$$

$$m(2,2)m(2,1) \quad H[0]$$

となります。そして、 $m(2,2)m(2,1) = [f_1, f_2, f_3, \dots]$ とすると (f_i は $M[1, i]$ の元)

$$f_1 = H[0] \text{ 程度の関数を元を持つ } M(1, 1)$$

$$f_2 : H[x_0] (H[a] \quad H[a \times 0] \text{ の変換})$$

$$f_3 : H[x_a] \quad H[x_a 0] \text{ の変換}$$

$$f_4 f_3 f_2 : H[x_0 0 0]$$

となることから、

$$m(2,2)^2 m(2,1) \quad H[1]$$

$$m(2,2)^3 m(2,1) \quad H[2]$$

$$m(2,2)^4 m(2,1) \quad H[3]$$

と計算されます。このように、 $m(2,2)$ が $H[a]$ の a に 1 を足す効果を持つので、

$$m(3,2)m(3,1) \quad H[0]$$

となります。

以上の計算を繰り返すと、

$$m(1,2)m(1,1) \quad H[0]$$

$$m(2,2)m(2,1) \quad H[0]$$

$$m(3,2)m(3,1) \quad H[0]$$

$$m(4,2)m(4,1) \quad H[0]$$

となり、

$$F_6(x) = m(x,2)m(x,1)(x) \quad H[0]$$

と計算されます。

8.4 Veblen 関数

【定義】 Veblen 関数

Veblen 関数 $\varphi(\alpha)$ は以下の様に定義される。

$$\begin{aligned} \varphi_0(\alpha) &= \omega^\alpha \\ \varphi_{\beta+1}(\alpha) &= \text{「 } \varphi_\beta(\alpha) = \gamma \text{ となる } \gamma \text{ のうち } \beta \text{ 番目のもの} \\ \varphi_\beta(\alpha) &= \text{「すべての } \gamma < \alpha \text{ に対し } \varphi_\beta(\gamma) = \alpha \text{ となる } \gamma \text{ のうち } \beta \text{ 番目のもの (} \beta < \text{limit) } \\ \varphi_0(0) &= 1 \text{ を満たす最小の順序数は Feferman-Schütte 順序数と呼ばれ、 } \varphi_0(0) \text{ と書かれる。} \end{aligned}$$

Veblen 関数について、巨大数スレッド 7-173 には以下の様に説明されています。

Veblen 関数の定義は、 $\varphi_{\beta+1}(\alpha) = \text{「 } \varphi_\beta(\alpha) = \gamma \text{ となる } \gamma \text{ のうち } \beta \text{ 番目の数} \text{」}$ ということですが、これは、「ある順序数 α より小さい数に φ_β , $\varphi_{\beta-1}, \dots$ を何回適用してもやはり α より小さい」という条件を満たす γ のうち β 番目のものと見ることができます。「 γ 」の条件は、「 γ より小さい数は $\varphi_\beta, \varphi_{\beta-1}, \dots$ という演算について閉じている」と言い換えることもできます。もう少し具体的な例で考えてみると、 $\varphi_0(\alpha) = \omega^\alpha$ より小さい数はいくら足し合わせても $\varphi_0(\alpha) = \omega^\alpha$ より小さな数となります。例えば、 $\alpha = 0$ のときは $\varphi_0(0) = 1$ より小さい数は 0 しか存在せず、0 をいくら足し合わせても 1 より小さいことから、1 が「その数より小さい数は φ_0 について閉じている」ような 0 番目の (最初の) 数ということになります。 $\alpha = 1$ のときは、 $\varphi_1 = \omega^{\omega^\alpha}$ より小さい数 (つまり自然数) をいくら足し合わせても φ_1 より小さいということになります。これを逆に考えると、自然数をいくら足し合わせても到達できないような最小の数を $\varphi_0(1) = \omega^{\omega}$ として定義していることとなります。この考え方で $\alpha = 2$ としてみると、 $\varphi_0(2)$ は 2 番目の数であるため 1 番目の数である $\varphi_0(1)$ より大きな数となります。なので、自然数や ω をいくら足し合わせても到達できないような最小の数を考えると、それは $\omega^{\omega^{\omega}}$ の収束先である $\varphi_1(0) = \omega^{\omega^{\omega}}$ ということになります。

$\varphi_1(0) = \omega^{\omega^{\omega}}$ で考えると、 φ_0 より小さい数に φ_1 や φ_0 を何回適用しても $\varphi_1(0)$ より小さいということになります。逆に考えれば、0 に φ_1 や φ_0 を何回適用しても到達できないような数のうち、最小のものを $\varphi_1(0) = \omega^{\omega^{\omega}}$ と定義していることとなります。 $\varphi_1(1) = \omega^{\omega^{\omega^{\omega}}}$ は、 $\varphi_1(0)$ 以下の数 (φ_0 も含む) に φ_1 や φ_0 を何回適用しても到達できない最小の数ということになります。

このように、 φ_0 だけでは到達できない数を φ_1 で定義し、 φ_1 と φ_0 だけでは到達できない数を φ_2 で定義し、 φ_2 と φ_1 と φ_0 だけでは到達できない数を φ_3 で定義し、...とやっていって、最終的には Veblen 関数を何回使っても到達できない最小の数を $\varphi_\beta(0)$ と定義していることとなります。

Veblen 関数とこれまでに出来た順序数との関係は、以下の様になります。

$$\begin{aligned} \varphi_0(0) &= 1 \\ \varphi_1(0) &= \omega^{\omega^{\omega}} \\ \varphi_1(1) &= \omega^{\omega^{\omega^{\omega}}} \\ \varphi_2(0) &= \omega^{\omega^{\omega^{\omega^{\omega}}}} \end{aligned}$$

$H[\varphi_\beta(0)]$ は、非常に大きな関数となります。Hardy 関数、順序数やチューリングマシンの概念を使わずにこの大きさの関数を作る事は大変難しく、そのため、巨大数スレッドにおいて現在 $H[\varphi_\beta(0)]$ への挑戦がされているところです。

8.5 拡張 Veblen 関数

Veblen 関数を拡張した 関数が、巨大数スレッド 7 の中で、以下の様に定義されました。

【定義】拡張 Veblen 関数

$= \lim_n$ と書いたとき、 α は極限順序数、 $\beta_n = \beta$
 が極限順序数のとき $\alpha + \beta = \lim_n \alpha + \beta_n$

の定義 (Veblen 関数の拡張)

$$(\dots, [\alpha, 0], \dots) = (\dots, \dots)$$

$$(\dots) = 1$$

$$([0, \alpha + 1]) = \lim_n \beta_n$$

ただし $\beta_0 = 0$, $\beta_{(n+1)} = \beta_n + ([0, \beta_n])$

$$(\dots, [\alpha + 1, 1], [0, \alpha + 1]) = \lim_n \beta_n$$

ただし $\beta_0 = (\dots, [\alpha + 1, 1], [0, \beta_n]) + 1$, $\beta_{(n+1)} = (\dots, [\beta_n, \beta_n])$

$$(\dots, [\alpha + 1, \alpha + 1]) = \lim_n \beta_n$$

ただし $\beta_0 = 0$, $\beta_{(n+1)} = (\dots, [\alpha + 1, \beta_n], [\beta_n, \beta_n])$

0 のとき

$$(\dots, [\alpha, \alpha + 1], [0, \alpha + 1]) = \lim_n \beta_n$$

ただし $\beta_0 = (\dots, [\alpha, \alpha + 1], [0, \beta_n]) + 1$, $\beta_{(n+1)} = (\dots, [\beta_n, \beta_n], [0, \beta_n])$

が極限順序数のとき

$$(\dots, [\alpha, 1], [0, \alpha + 1]) = \lim_n (\dots, [\beta_n, 1], [0, (\dots, [\beta_n, 1], [0, \beta_n]) + 1])$$

$$(\dots, [\alpha, \alpha + 1]) = \lim_n (\dots, [\beta_n, \beta_n], [\beta_n, 1])$$

が極限順序数のとき

$$(\dots, [\alpha, \beta_n]) = \lim_n (\dots, [\beta_n, \beta_n])$$

$$(\dots, [\alpha, \beta_n], [0, \alpha + 1]) = \lim_n (\dots, [\beta_n, \beta_n], [0, (\dots, [\beta_n, \beta_n], [0, \beta_n]) + 1])$$

巨大関数の定義 (F は Hardy Function)

$$f(x) = Fx \quad \text{ただし } \beta_0 = 0, \quad \beta_{(n+1)} = ([\beta_n, 1])$$

小さい値について計算してみると、こうなります。

$$f(0) = F0 = 1$$

$$f(1) = F[[0, 1]](1) = F1 = F[0](1) = 2$$

$$f(2) = F[[[0, 1], 1]](2) = F[[([0, 1], 0), [2, 1]]](2) = F[[2, 1]](2) = F[[2, 0], [1, [2, 0], [1, 0]]](2) = F[[1, ()]](2) = F[[1, 1]](2) = F[[1, 0], [0, ([1, 0], [0, 0])]](2)$$

$$= F[([0, ()])](2) = F[([0,1])](2) = F2 = F[1]F[1](2) = F[1]F[0]F[0](2) = F[1](4) = F[0]F[0]F[0]F[0](4) = 8$$

8.6 Taranovsky の順序数記法

Dmytro Taranovsky が、以下のページで大きな順序数の記法を定義しています。

<http://web.mit.edu/dmytro/www/other/OrdinalNotation.htm>

この文章によれば、順序数記法を以下の様に定義できます。

1. A General Notation
2. An Ordinal Notation
3. A Stronger Ordinal Notation
4. Second Order Arithmetic
5. Beyond Second Order Arithmetic

最初の An Ordinal Notation の段階で、Veblen 関数、Bachmann-Howard ordinal、proof-theoretical ordinal、Transfinite Recursion、Monotonic Induction の段階まで記述でき、さらにその先まで記述できるようになります。

たろう氏が、Taranovsky の順序数記法について、収束列の定義を試みています。

8.6.1 多変数 C_0

Taranovsky の順序数記法を元に、たろう氏が作成した順序数から順序数への多変数関数です。「0 とこの関数だけで、() 未満のすべての順序数を作ることが出来る」とのことです。

【定義】多変数 C_0

α : 0 個以上の 0
 X : 0 個以上の 0 以上の順序数
 a, b : 順序数
 B : 極限順序数 (B_n : 収束列)
 として、
 $C_0(\alpha, a) = a + 1$
 $C_0(X, b + 1, a) = \lim$ 初項 a / 2 項目以降 $C_0(X, b, 1$ 個前の項)
 $C_0(X, b + 1, 0, \alpha, a) = \lim$ 初項 0 / 2 項目以降 $C_0(X, b, 1$ 個前の項, α, a)
 $C_0(X, B, \alpha, a) = \lim C_0(X, B_n, \alpha, a)$

大きさは、以下の様に計算されています。

$$C_0(a, 0) = a$$

$$C_0(1, 0, 0) = 0$$

$$C_0(2, 0, 0) = 0$$

$$C_0(a, 0, 0) = a(0)$$

$$\begin{aligned}
C_0(1, 0, 0, 0) &= {}_0 = (0) \\
C_0(1, 0, a, 0) &= {}_a \\
C_0(1, 1, 0, 0) &= \dots {}_0 = () \\
C_0(1, a, 0, 0) &= (\cdot a) \\
C_0(2, 0, 0, 0) &= ({}^2) \\
C_0(2, 1, 0, 0) &= ({}^2 +) \\
C_0(2, 2, 0, 0) &= ({}^2 + \cdot 2) \\
C_0(2, 3, 0, 0) &= ({}^2 + \cdot 3) \\
C_0(2, a, 0, 0) &= ({}^2 + \cdot a) \\
C_0(3, 0, 0, 0) &= ({}^2 \cdot 2) \\
C_0(4, 0, 0, 0) &= ({}^2 \cdot 3) \\
C_0(, 0, 0, 0) &= ({}^2 \cdot) \\
C_0(1, 0, 0, 0, 0) &= ({}^3) \\
C_0(1, 0, 0, 0, 0, 0) &= ({}^4) \\
C_0(1, 0, 0, 0, 0, 0, 0) &= ({}^5) \\
\lim C_0(1 \text{ が } n \text{ 個}) &= ()
\end{aligned}$$

8.6.2 2重リスト C_0 以上

さらに、2重リスト C_0 、多変数 C_1 などが定義されています。まだ著者の理解が追いついていません。

8.7 その他の関数

巨大数スレッドで提案、議論された様々な関数、数を紹介します。著者自身が、まだ定義の理解と議論に追いついていないところもありますので、この章では簡単に紹介するに止めます。詳しくは、巨大数スレッドを参照してください。ここに記述したもの以外にももまだありますが、ここでは特に活発に議論がされているものを紹介します。

8.7.1 ナゴヤ関数

巨大数スレッドで提案された関数です。Hardy 関数の引数を順序数のリストに拡張することと、対格化を使った拡張を基本的なアイデアとしているようです。

8.7.2 アルカ数

ルート 41 氏が巨大数スレッド 8 で提案し、議論されています。アルカ数、マダ・アルカ数、マダ・アルカオメガ、マダ・アルカグラハム数、アス・アルカオメガといった数が定義されています。

8.8 帰納関数の大きさ比較

表 8.1: 帰納関数の大きさ比較

Hardy 関数	相当する関数
$F[]$	<i>Knuth</i> の上矢印 $3(\quad n)3$
$F[\quad 2]$	n 個の n からなる Conway のチェーン, F_3 の $s(3)$ 変換
$F[\quad 3]$	チェーンの回転 $n(3, 3, 3)$, F_3 の $s(4)$ 変換
$F[\quad]$	多変数アッカーマン, F_3 の $ss(1)$
$F[\quad]$	2重リストアッカーマン, F_5 の $m(5)$
$F[\quad 0] = F[C_0(1, 0, 0)]$	ヒドラゲーム, 多重リストアッカーマン, $F_5(x)$
$F[\quad 0] = F[C_0(2, 0, 0)]$	$F_6(x)$
$F[\quad 0] = F[C_0(1, 0, 0, 0)]$	
続く...	

第9章 計算不可能な関数

9.1 ビジービーバー関数

9.1.1 チューリングマシン

9.2 ふいつしゅ数バージョン4

【定義】ふいつしゅ数バージョン4

[1] 関数 f から関数 g への写像 $s'(1)$ を以下で定める。

関数 f を神託 (oracle) として持つチューリングの O -machine を考え、このマシンによるビジービーバー関数を g とする。すなわち「チューリングマシン+関数 f 」のマシン n ステートでセッティング可能な1の数の最大を $g(n)$ とする。

[2] $s'(n)(n > 1)$ および $ss'(n)(n > 0)$ を、 F_3 と同様に定める。すなわち、

$$s'(n)f := g; g(x) = [s'(n-1)]^x f(x) (n > 1)$$

$$ss'(1)f := g; g(x) = s'(x)f(x)$$

$$ss'(n)f := g; g(x) = [ss'(n-1)]^x f(x) (n > 1)$$

[3] ふいつしゅ関数 $F_4(x)$ を以下のように定める。

$$F_4(x) := ss'(2)^{63} f; f(x) = x + 1$$

[4] ふいつしゅ数 $F_4 := F_4^{63}(3)$ とする。

9.3 ビジービーバーのHardy的拡張

たろう氏により、ビジービーバーのハーディ関数的拡張が以下で定義されました。

【定義】ビジービーバーのハーディ関数的拡張

$$B[0](n) = n$$

$$B[a+1](n) = s'(1)(B[a])(n)$$

$$B[A](n) = B[A_n](n)$$

ここで、 $s'(1)$ はふいつしゅ数バージョン4の定義と同じ。

この時、 $F_3(n) = F_3^{+1 \times 63}(n)$ であることから、 $F_4(n) = B_4^{+1 \times 63}(n)$ となります。

すなわち、大きな順序数からハーディー関数を使って大きな計算可能関数を作ったのと同様の手続きで、大きな順序数から $B[A]$ を用いて大きな計算不可能関数を作ることができます。前章の巨大順序数の議論をそのまま当てはめ、 $B[\omega_1]$ といったような大きな関数を作ることができます。

関数 $B[A]$ は、計算不可能ではありますが厳密に定義されている関数です。

9.4 帰納的でない順序数と Hardy 関数

ω_1^{CK} を Church-Kleene ordinal として、 $H[\omega_1^{CK}] = \text{BB}$ 説
さらに、 $CK[A]$ など強めていく。