A Hitchhiker's guide to Lambda (λ)

人七 厩

2009-4-11(Sat)

Contents

Abstract

以前私は λ 計算は数学の基礎を作り直そうとして考え出されたという話を聞いた．そしてそれに興味を持った．λ 計算に関しては既に様々な解説がある．私の場合，Wikipedia が良いスタートであった．しかし実際にこの計算をしてみようとするとすぐに壁にあたってしまった．これらの Web Page では,「自分で確かめてみるとよいだろう」という部分があり，それがわからなかったのだ．考えた末にようやく一番単純な部分が理解できた．これはそのメモである．

1 まえがき

この文章は私が 2008-9-8 から 2009-4-11 にかけて blog [?, ?] に掲載したものの日本語版を base に多少手直ししたものである．

2 λ 計算とヒッチハイカーズガイド

ブログを書くことなどないだろうと思っていたが，友人のブログが一週間お休みになってしまったのを機会に自分の興味あることを書いてみることにした．

今回のネタは λ 計算というものだ．実はいったいこれがなんなのかが知りたくて，Wikipedia を見てみた．ところが，実際の計算を理解するまでに一週間かかってしまって，友人のブログはもう再開されている．λ 計算はとても特殊なテーマであるし，友人のブログが面白いのでそっちを読めばもう書かなくてもいいかなと思ったが，一週間かかってわかったことを自慢したくなったので書いてみよう．しかし，コメントに「そんな簡単なことを一週間も考えないとわからないとは，愚かものめ」，と書かれるかもしれないと思うとちょっと勇気が必要である．

とにかく何がとてつもなく日常世界と違うことに出会ったら，そのガイド本を開くのは一つの手である．有名なガイド本には地球の歩き方とか，銀河ヒッチハイカーズガイドとかがある．私は銀河ヒッチハイカーズガイドという小説のファンなので λ 計算のヒッチハイカーズガイドを書いてみようと思う．

銀河ヒッチハイカーズガイドによれば，銀河ヒッチハイカーズガイドという本は銀河で最も売れている本の一つである．しかし，それがいったいどの言語で書かれているかは明らかにされていない．売れている本ということは，どうやら，銀河のどの人達もそれを読むことができるようだ．ただし，光の感覚器官が無い人達はいったいどうやってガイドを読むのだろうかという疑問は残る．

ガイド本は人工頭脳を内蔵しており，各星人は Babel fish を使って理解するという方法もあるかもしれない．Babel fish というのはこの小説の中でテレパシーを使って翻訳をしてくれる便利な魚であるが，魚なのでよく間違える．(この小説から名前をとった翻訳エンジンもある．)

しかし，小説の主人公 Arther は確か Babel fish を入手する前に既にガイドを読んでいたようだ．とすると，何らかの翻訳機が入っているのであろう．銀河中の各星人向けの翻訳があるのではというふうに考える人もいるかもしれないが，ガイド社が各星人向けに翻訳するような手間をかけるはずがない．ガイド社はかつて時間と空間を越えてたった一冊の本を全ての平行宇宙に売ろうとした会社である．

翻訳装置ができるとしても，その本の native な言語というものは多分あるだろう．内部表現というやつである．本が最初に書かれた時に使われた言語かもしれない．どんな言葉が使われているかというかは想像することもできないが，論理などは言語にあまり依存しないでようにすることもできる．しかし，どんな論理も人間に理解できるようにするには，どこかで自然言語とのインターフェイスをとらなくてはいけない．論理自身は最初にいくつかの定義が必要である．たとえば真とか偽とは何かとか，数字の 1 は何かというようなことである．

まずは λ 計算がどんなことを考えているのか，その内部表現は何なのかを続きで書いてみようと思う．

3 λ 計算と函数

λ 計算は計算とつくだけあって，数も計算できるが，その方法は特殊である．λ 計算は数学を再構築しようとして考え出されたものであるので，数そのものが何かということも考えているのである．

実際，数というものをつきつめて考えていくと，よくわからなくなってしまう．たとえば数を知らない小さな子供にどうやって数というものを教えたら良いのだろうか．ただ，数についてのいくつかの性質は挙げることができそうだ．まず，どう読むかはどうでも良いことである．ガイドにおいて地球という既に失なわれて久しい星には多数の言語が乱立していた．たとえば英語とかドイツ語とか日本語とかいうものである．面白いことだが，どの言語にもどうやら数というものはあるようだ．そしてそれぞれ 1 2 3 (one two three, ein zwei drei, いち にさん) のようにいろいろな読み方がある．しかしどう読んでもその表すことは同じはずである．ガイドの内部では 1 2 3 は (ニシン サンドイッチ ニシンのサインドイッチ) と読んだりするかもしれない．しかし，しつこいようだが，結局どう読むかはどうでも良いことである．つまり読み方とは関係ない何かが，数の本質にはあるはずだ．ガイドの内には「何か」がつまっていて，読者に応じてその「何か」が読者に理解できるような形に変換されて提示される．一つ前の章でガイドの内部表現と言ったのは，ガイドの内容そのものである．何語で書かれているかは本質的にはどうでも良いのだ．

λ 計算とは函数について考える数学である．函数または関数 (function [1]) という便利な考えが数学にはある．何かを入れると何かが出てくる自動販売機のようなものだ．普通，1 アルタイルドルは何シリウス円なのかなどということを教えてくれる箱のようなものと考えることが多い．ガイドの中にもそういう函数があるのではないかと思う．

別に函数は箱である必要はないのであるが，私の場合，最初にたまたま自動販売機の例えで教

[1] 本来は函数と書くようであるが，函の文字が常用漢字にないので同音の関を使うようだ

わったせいか，はたまた函 (はこ) という文字があたっているからかわからないが箱を思いうかべてしまう．このような函数を地球では標準的に f とか g とかと教えるが，教える方も教わる方もそれが何かはあまり気にしなかったり，結局よくわかっていないようだ．コンピュータ学者はわかっていないことを素人に知られるとたいへん困るので，函数というような既にポピュラーになってしまった言葉を使わずに，いろんな言葉を使う．

λ 計算というものもたいていの計算機学者は素人を誤魔化すための言葉として使っている.「不動点定理むにゃむにゃ」「計算可能性なむなむ」とかいう大学の教授がいたら，騙されないようにしないといけない．特にスーツを着ていて偉そうなのはあやしい．

でも λ 計算は函数というのが何かを考える時には便利なこともある．ま，私もわかってはいないので,「チャーチロッサー性がむにゃむにゃ」とか言ってここで話を終わりにしても良いのだが，ものを書こうなどという人間の性で，ちょっとは知っていることを自慢したかったりする．とりあえずガイドの物語で有名なロボット，マービンがやってくるまでは話を続けよう．ところでマービンは最低の気分を持っている宇宙最高のロボットである．シリウスサイバネティクス社が300 人分の頭脳を持った天才ロボットを試作したところ，天才と正気は両立しない，つまり天才は正気ではない，ということが判明したのでマービン型のロボットの生産はキャンセルされてしまった．それがマービンをさらに特別なものにしている．

私の大好きなマービンがようやく紹介されたので，今日はここまでにしよう．

4　番外 1

前回するどいコメントがついた．壊れた自動販売機だったらどうするのかということである．しかし,「壊れたというのはどういうことか」をまずはっきりさせないといけないだろう.「壊れているというのは壊れているということだ」，と言う人もいるだろう．しかし数学者にはそんな言葉は通じない．数学者には常識はないのか，そうかもしれない．もちろんそれには理由がある．

私が数学を趣味にしているのは，主に数学を使って機械 (計算機) に仕事をさせて楽をするためである．したがって，私個人にとっては機械が解釈可能であるような考えでないとそんなに面白くないのである．数学というのは機械に解釈できる部分が多い，だから私には有用だし，楽しみがある．情報を探してこいとか，家から駅までの最短距離を教えてくれとか，安いチケットを探してこいとかいうのも，機械が解釈できる形になっていなくては何もできない．壊れているというのがはっきりしないというのは，それを機械にどう解釈可能な形で指定するかというのがはっきりしないということである．

たとえば，入力しても何もでてこないような函数は壊れているのか，それとも予想できないものがでてくるのが壊れているのか，あるいは何を入れても同じものしか出てこなかったらそれは壊れているのだろうか．というように壊れていると考えられることはいくつもある．このように何をもって壊れているのかを指定できない場合には現在の機械には解釈が難しい．

また，壊れているという言葉はある意味主観的なものでもある．ある種類のチップはその内部の情報が重要であるがために，中を覗こうとすると二度と読み出せなくなる機能がある．つまり，ある条件で壊れるように作られている，とも言える．たとえば，クレジットカードの暗号を保存しているチップや，コピーのガード機能のチップにはこういう機能があるものがある．二度と内部の情報が読み出せないということを「壊れる」という表現を使う人がいる．しかし，それは実は設計者にとっては正しい動作であって，設計者にはまったく壊れていないのである．設計者にとってそういうチップが壊れているという意味は，情報が漏洩しそうなときに壊れないチップが壊れているのである．こういうものは人間には理解が簡単であろうが，現時点の機械にはなかなか難しい．誰かにクレジットカードを盗まれた人がいたとしよう．その人はチップが他人には使えなくなることを期待するだろう．もし，チップが壊れていて，盗んだ人が秘密の情報を読みだすことができて，カードが利用されてしまったら，クレジットカードの所持者はそのチップが「壊れていたから壊れなかった」として訴訟を起こすかもしれない．

λ 計算は計算が何かを考えるものであるから当然こういうような函数についても考えているのである．簡単に言えば，上記の壊れているというものの表現ができないのであれば，それはこの計算方法の限界を示すのであって，不完全なのである．ただし，どういうものが壊れているのかはちゃんと人間が定義する必要がある．数学者は面倒くさがりやだが，不完全なものは嫌いである．いかにさぼって完全なものを手にするのか，そう,「さぼり」ですら「完全なもの」を追及するのが数学者の美学である．だからそういうことも当然考えているのである．数千年の数学の歴史は数学の完全性に関しての理論で一区切りがつくほどである．しかしこのテーマは，ここで述べるには大きすぎるし，私のような単なる日曜数学が趣味の者には荷が重いので言及できないかもしれない．でも，機会があればどこかでこのことに関して少しは述べることにしよう．

5　λ計算 への導入

標準的な数学の本ではある数学的ことがらを説明するのに，定義，定理，証明，定義，定理，証明,,,のように話が進んでいく．そのようにすると話は(ある意味) すっきりするし，数学の抽象化というものが上手く働く．だからλ計算に関してもそのように話をすすめるのが標準なのだが，マービンに言わせればそれほど気が滅入ることはないだろう．抽象化されたものは応用が利くし，余計なことを言わないので単純さと美しさがある．つまり，的を射ているのである．

日本の刀はその美しさを刃に求め，余計な宝石で飾ったりしない．美しさはそのもの本質を追及することにあると考えるからである．美術館には刀や杖を宝石で飾りたてたものが多くある．そこに豪華な美を見ることも私にはできるが，本質と美しさを単純さの中に求めたものにも私は美を感じる．ある飛行機乗りが「全ての不要なものを取り去った時，完全性が達成される」と言ったという話を聞くと共感できるものである．そしてかつてスウェーデンの海に沈んだVasa [?,?] の話は私のお気に入りの一つでもある．これはプロジェクトマネージャーの王様が機能拡張を要求しすぎた例である．シリウスサイバネティクスのソフトウェアは顧客が欲しいとマネージャが思った機能を追加しすぎて使いものにならなくなったソフトが多い．顧客は単に機能縮小しても，安定したソフトが欲しかったりする．

数学の美しさはその抽象性と単純さにある．だから余計な背景や説明を除いてそのものだけを論じたくなるのはよくわかる．しかし，それはわかる人にはわかるということである．私はここに排他性を見る．

あるものの側面に,「ああ，美しいな」ということを発見することはとても嬉しいことである．そして時にその発見した美を独り占めしたくなるのも人情かもしれない．美が広く知られることによって大衆化し，皆が知っていると寂しく感じるということもあるのかもしれない．数学を愛する人の中には数学をますます純化し，抽象化してその美しさを求めてしまう者がいる．しかしそうなったものは本当にその的だけになってしまい，その背景にあったものや歴史を捨ててしまう．それは本質なのだから，それでいいのかもしれないが，親しみがなくなってしまう．私にとってはそのような純粋なものは純粋すぎるが故に理解した気にならないのである．

純化の過程で捨てられてしまったものも，私は気になってしまう性質である．なぜλ計算なんてものを考えるのか，ということはこの体系ができてしまった後では重要ではない．λ計算に何ができるかというのが結局より本質的なことである．数学ではそれそのものよりも，それが何ができるかの方が重要である．

ガイドのボゴン人のビジネスはまったくドライである．相手が何ができるかということこそが重要であって，それが誰かということはまったくどうでも良い．いや，彼らには親戚関係だって単なる関係でしかない．おばあちゃんが助けを求めても，孫たちは契約書がなければ指一本動かさないのである．ドライな会社のようなものである．新しい機能を実装できるかどうかが従業員に求められているものの全てである．もちろんそれこそがビジネスには重要なことではあるのだが，それを行う人間はある意味本質ではなく，どうでも良いのである．その会社の誰かがそれをできれば，誰がやってもかまわない．するとそれは誰がやったかという意味がなくなる．通常は会社の機能としか見られない．ただし，人間の場合には，機能のみを追及されてば，そこでの働きがいがなくなってしまう．なぜなら，機能が満たされれば誰がやっても同じだからである．そのような会社は人材を重視せず，最後には機能も失うことが多い．比較的長い学習期間の必要な職種，たとえば，Software development は最低でも数ヶ月，時には数年かかることがある．これは近年の Software が高度に複雑化しているからである．このような会社の manegement が社員はスーパーの棚の奥にあって安いものと取り換えが利くと思うようになると，おしまいである．

λ計算に何ができるかが重要であって本質であることは確かだが，ここでは余計なことを述べてみよう．まず，どうしてλ計算なんてものを考えようと思ったかである．λ計算だって人間の考えたものであるから，何か動機があるはずである．マービンもあれほど気が滅入っている存在は宇宙にはないほどであるが，そういうものを作りたくて作ったのではなかった．単に天才を何百人集めたら大天才になるだろうという考えで設計されたのだ．しかし，何百人分もの天才が集った脳が正気であるという保証はなかったこととか，天才はえてして紙一重であるという古くからの知恵を無視してしまった結果であった．λ計算もある目的のために考えられた，それが後から見ていかに変であっても，目的自身はそんなに難しいものではなかった．次回はλ計算の動機について書いてみよう．

6　λ 計算の動機

λ計算は Church さんと Kleene さんが 1930 年代にこれを考えたとWikipedia [?] (日本語 [?]) にはある．彼らは最初これを数学の基礎を築こうとして考えたとある．

ところで私は素人なのであまりここに書いていることを鵜呑みにしないようにお願いする．こ

れは私が理解する限りにおいての話である．この blog の元ネタは Wikipedia である．かっこ良く言えば，Wikipedia に inspire されて書いているのである．もし，あなたがこの Wikipedia の Page の内容をわかってしまったのであればこの blog よりも面白いものが沢山あるのでそちらをおすすめする．たとえば,「数学の基礎を築こうとして λ 計算は考えられた」と聞いて「なるほど，そうかそうか」，と思う人はこの blog を読む必要はない．しかし,「1930 年というと数学の歴史からするとかなり最近のことだ．どうしてこの時に数学の基礎なんてものを考えようとしたのだろう．いったい数学の基礎というのはなんなのだろう．ピタゴラスやアポロニウス，ユークリッドとかが数学の基礎を作ったりしたのではないのだろうか」とか思うのであればもう少し読んでもいいかもしれない．

数学の基礎というのは数学はどこから出発できるかということと，どういう出発ならば数学の体系に問題がないかという疑問に端を発っする．数学はある定義から出発する．これは人が決めるものである．まず，数学で最も基本的なものは何かと言うと数であろう．他にはある「命題」が真とか偽とかいう論理や，計算とは何かとかいうものも基本的である．こういうものはあまりにあたりまえすぎてかなり後の時代になるまで注目されなかった．

しかし，あたりまえのことに注目してどうするというのだろうか．数学者達はこの時，形式化ということについて考えるようになっていた．あたりまえに思える数であるが，世界中の人 (少なくとも数学者) が本当に同意できるものなのだろうか．たとえば，それは言語によって違うのかもしれない．日本語での 1 と英語の 1 は本当に同じものだろうか? そこで数というものから定義しなおすことを考えた．ところで数を定義する時には数は使えない．もし使えると,「数とは数である」と書けば良くなってしまうからである.「愛とは愛である」「自意識とは自意識を自意識できるものである.」は真かもしれないが，こういう定義では数学の基礎としては使えないのである．数というものを数を使わないで定義できるのか? ここまでくると数学者は偏執狂かという感まである．

私がこの考えを面白いと思うのは，これが十分に形式化されているので，機械でそれを作って実行できるという点である．計算する機械を作ろうとすると，それをなんとかして物質 (鉄とか水とか石とか) で作らなくてはいけない．Zaphod ならば 2 つの頭で「わかるだろう，1 とか 2 とか 3 だよ，そういうのをとりあえず計算してみせる機械をちゃちゃっと作ってみてくれよ」と言うかもしれないが，どうやって 1 とか 2 を示せば良いのだろうか．

数というのは驚くほど抽象化された概念である．抽象化されているというのは，非常に応用範囲が広いということである.「地球と人間とパンと信号機と Zaphod の頭と教会の共通点は?」という問いに,「どれも数えられるものだ」という答えは正しい．繰り返しになるが，私には数をどうやって子供に教えたら良いのか見当がつかないのである．これは高度に抽象化された考えである．

λ 計算の動機は数学の基礎を築くためにあった．というのは，それまであたりまえと考えられていた数や計算を形式化して 1 から考え直すということであったのだ．当時の数学者には数学のシステムの正しさを単純なルールだけで示すことが目的であったようだが，このルールは十分単純化されており，機械でも作れる．私が λ 計算に興味を持ったのはこの「機械としても作れる」という所である．λ 計算が現代の計算機言語の基礎理論の一つであるのはそういうわけである．

次回は数 (自然数) を 1, 2, 3 などを使わずに定義してみよう．

7 λ計算 における自然数

7.1 自然数の定義

自然数を定義したのはPeano [?] さんという人である．この方は自然数の性質を述べたのであって，こうやって自然数を定義しようとしたようではないようだ．数学の書き方に慣れていないととっつきにくいのであるが，そんなに難しいことが述べられているわけではない．次の 5 つの定義はWikipediaから転載した．

1. 自然数 0 が存在する
2. 任意の自然数 a にはその後者 (successor), $\mathrm{suc}(a)$ が存在する ($\mathrm{suc}(a)$ は $a+1$ を意味する)
3. 0 はいかなる自然数の後者でもない (0 より前の自然数は存在しない)
4. 異なる自然数は異なる後者を持つ : $a \neq b$ のとき $\mathrm{suc}(a) \neq \mathrm{suc}(b)$ となる
5. 0 がある性質を満たし，a がある性質を満たせばその後者 $\mathrm{suc}(a)$ もその性質を満たすとき，すべての自然数はその性質を満たす

まず，数には最初の数というものがあると定義する．ここでは 0 と書いているので数の 0 のように見えるが,「何か」であればかまわない．ここで「何か」というと「何かって何?」と尋ねる方もおられよう．しかし，実際に「何か」であれば良いのである．0 というのも最初の「何か」であって，1 でも 42 でも x でも良い．あるいは，

日本語で「零」と書いても良いし,「zero」 でも「Null」でも良いのである．本当に「何か」であれば何でも良いのである．このような考えは時に受け入れてもらえないこともわからないではない．ここでは自然数を定義したいので，既に 0 という自然数があるように思ってもらっては困るのである．(また，0 は自然数としないことも多い) そこで 0 という記号を使って最初の数を定義するのである．個人的には私は x の方が「何か」という感じがあって良い気がする．また，Peano 自身の公理では最初の数は 1 であったにもかかわらず，Peano の公理と言うと 0 を使うようである．

定義というのはゲームのルールのようなものでそれが何故かということは考えない．これは数学に慣れていない人には難しいことだと思うが，サッカーで「ゴールキーパ以外はボールを手で触ってはいけない」というルールと同じである．何故そうなのか理由を言えと言われてもそれがサッカーのルールなんだとしか言えない．チェスや将棋，囲碁のルールとも同じである．つまり唯一真実のルールというものは存在しない．ゲームの数だけ，スポーツの数だけルールがある．数学でも同じであり，いくつの異なったルールを基礎にした数学が存在する．数学ではとんでもないルールを作っても，一貫性さえあれば問題はない．しかし，とんでもないルールの数学は普通面白くない．面白いかどうかはまったくもって人の主観である．時々誤解があるようだが，ここには宇宙の真理とかはまったく関係ない．しかし，面白いことに良く作られた数学のルールは宇宙の真理に迫ることができる．これは数学の素晴しいところである．それは良く作られたスポーツやゲームのルールはそのスポーツやゲームを面白くする所にも似ている．唯一真実のルールがあるというのは，宇宙には一つのスポーツしかなく，世界には一つのゲームしかないというようなものである．新しい定義を作れば，新しい数学ができるが，面白い数学を作るのは大変むずかしい．適当なルールをつくれば新しいスポーツができるだろうけれども，面白いスポーツを作り出すのは難しいことに似ている．

最初の自然数からは次の自然数というものを作ることができるというのが二番目の定義の言うところである．この操作をする函数をあるものの後者を生むので「後者函数」と言う．これは $+1$ という演算を定義していると思って良いだろう．もし 0 があれば次の「何か」(0 の次なら 1 と書くこともあるが，別に 0 と違う「何か」ならばなんでも良い) が生成される．ところで，1. 最初の数がある．2. 次の数を作ることができる．という 2 つのルールで自然数の定義ができそうなのだが，これではまだ足りない．

3 番目の定義は，後者函数を繰り返す，つまり自然数の足し算 (+1) を繰り返していっても最初の数に戻ることはないという意味である．つまり x+1+1+1... としていったらある時突然 0 に戻ることはありえないよ．という意味である．そういう数学 (modulo) もあるのだが，自然数の定義では考えない．たとえば，このようなものでない数学としては $12+1=1$ とか $31+1=1$ となるものがある．そんな馬鹿な，と思うかもしれないが，これは日付けの表現である．なんと 12 月 31 日の次は 1 月 1 日に戻ってしまうのだ．月を見れば $12+1=1$ であり日付けだけ見ると $31+1=1$ なのである．時間の表現もそうである．23 時 59 分の次は 0 時 0 分である．そんなのは計算ではないと言うなかれ．計算機の中のカレンダーはもちろんそういう計算をしているのである．だからそういう数学だってある．$1+1=2$ というのはそんなに普通ではないのだ．これを一番簡単な数学の例として挙げる人がいるが，そんなに簡単ではないと思う．

4 番目の定義は同じ数の次の数は同じであるという意味である．逆に言えば違う数の次の数が同じになることはない．カレンダーの例ではこれは成立していないことがおわかりだろうか．カレンダーの場合，31+1 = 1 (一月), 28+1 = 1 (二月)，30+1 = 1 (四月) など違う数の次の数が同じになっている．自然数ではこういうことは起きないというのがこの定義の言う所である．そろそろマービンがあたりまえすぎて嫌になってくることだと言いそうだ．最後の定義は数学的帰納法に関係する．こうやって作られたものは全部自然数とするというものである．

マービン「ところでこれは志賀浩二先生の説明にとっても似てますね．ちゃんと参照しておいた方がいいんじゃないですか.」確かにそうである．確か数学が育っていく物語であったと思うのであるが，残念ながら手元に本がない．私は志賀先生の本が好きで横浜国立大学の公開講座に毎土曜日に電車で 2 時間かけて通ったことがある．すばらしい講義であった．日本を離れる時にはこの講義を聞きに行くことができなくなることが残念でならなかった．

また Peano の定理というと私は老子と Wittgenstein を考えてしまう．ガイドファンにはたまらない偶然で，老子の「42 節」は「道は 1 より生じ，1 は 2 を，2 は 3 を，3 は万物を生んだ」とはじまる．また，私が Wittgenstein の言ったことを理解しているはずもないが，言葉は世界の写像にすぎないというのはその通りだと思う．λ 計算は言葉をまた機械に写像しなおすことにより，論理を世界に還元する．人間が言葉でどう言うかではなく，機械という世界の構成要素が，自然数と同型の動作を見せることができるというだけで私には満足である．

次はついにこの公理を λ で示すことを考えよう．

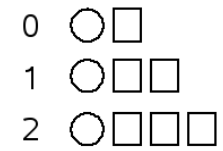

Figure 1. An example representation of Peano's number

Figure 1: An example representation of Peano's number

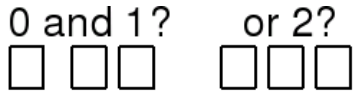

Figure 2. we need a delimiter

Figure 2: Necessity of delimiter to distinguish the numbers

7.2 チャーチ数 (1)

前回は Peano がどうやって自然数を定義したかの話だった．λ 計算でも実は同じ方法で自然数を定義するので Peano の話をしたのだ．形式化自体は証明を厳密にするということにかかわってくるので，数学者にはそちらの方が重要であるようだ．しかし，形式化によって厳密さが増した結果「数なんてもの位，わかるだろう，適当にやってくれよ」という部分がなくなる．あたりまえのようなことも全部定義されるからである．こうなるとその副作用で機械で作ることも可能になる．コンピュータを作ることができるのだ．私にとって形式化が面白いのはこの部分である．

実際どのように機械で作るのかはいろいろな方法がある．パスカルの計算機 (Pascaline [**?**]) や Charles Babbage の (differencial engine [**?**]) のような歯車方式もあるだろう．ここでは Peano の定理に近いものを考えよう．

マービンに言われるまえに言っておくが，これは佐藤雅彦，桜井貴文両先生の「プログラムの基礎理論」にある方式，だったと思う．(本が皆実家にあるというのは不便なものだ．ところでこの文章は佐藤先生が最初に実装された SKK という入力方法で書いている．) 佐藤先生の授業はタフな授業である．聞いているだけでは，少なくとも私には，絶対にわからない．しかし，質問をするとどんなに馬鹿げていることでもわかるまで答えて下さる．ところが，あまりにタフな授業なので何を質問して良いのかがわからない．質問できるレベルまで予習をするのは大変であった．

まず，数であることを示す記号として丸を使う，そして Peano でいう 0 を四角で示すことにする．次の数は 0 を繰替えすことで示そう．Figure 1 を参照のこと．

ここでは 1 とか 2 は使っていない．というのも，定義ではそういうものがないからだ．現在の 1 とか 2 の記法は 10 進法に関係するのでそんなに本質ではない．ローマ数字のような方式でも数は示せるし，メソポタミアの方式や，漢字という手もある．これらは数そのものをより人間に便利になるようにしてあって，自然数そのものはこのように 0 と数字の区切りがあればで示せるのである．その 0 も別に何語で書いても良いのでここでは四角を使う．

四角が一つだから 1 にしたいという考えもあると思う．それは確かにそうなのだが，0 を示すにも何か記号が必要だ．「0 は何もないことがある」ことを示す記号である．ここでは四角しかないのでこうなってしまった．ここでは数のようなものを作って，それを実際の数に map することになるので，どこからはじめるかはあまり問題ではない．自然数は 1 からはじめても良いし，42 からはじめても良いのだ．ただ直感的には Figure 1 方式では 1 からの方が良いとは私も思う．

また，本来四角だけで済むのに丸を導入した理由は次回に示そう．

7.3 チャーチ数 (2)

前回丸は数を示す記号といったが，Peano の定義にはでてこない．定義されたのは Zero だけであり，Zero は四角を使ったのであった．機械に 2 つの数を教える時に四角を使っているだけでは Figure 2 のようにどこに区切りがあるかわからないので，丸を使って区別しているのだ．空白記号 (スペース) を使えばいいではないかというかもしれない．もちろんそうだが，スペースという記号もやはり機械に教えなくてはいけないのだ．たとえて言えば数字の 0 のようなものが必要なのだ．0 というのは「何もないということがある」ことを示している．何も書かないとあるのかないのかまったくわからないが，0 を書くことによって「0 がある」ことを示すことができる．0 は「無いことを示すことができる」のでまったくすばらしい．ところでスペースと言えば日本語には単語を区切るスペースがない．だから日本語処理は単語を区切る形態素分割から初まる．スペースは何もないことを示す文字である．これも 0 同様にすばらしい発明である．老子で言えば無用の用であろうか．

λ 計算での数，チャーチ数は Figure 1 と同じように定義される．λ 計算では函数ということを示すために λ を頭につける．全ての数が λfx で始

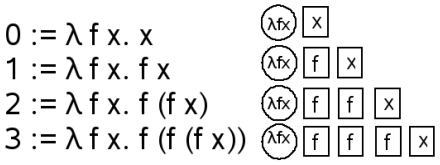

Figure 3.Church numerals

Figure 3: Church numerals

まっているのは，これは函数であって，f と x が関係しているということを示している．詳しい話は後にもう一度することにして，ここではチャーチ数を λ で書いたものがどのようなものかを示しておこう．

$$\begin{aligned} 0 &:= \lambda fx.x \\ 1 &:= \lambda fx.fx \\ 2 &:= \lambda fx.f(fx) \\ 3 &:= \lambda fx.f(f(fx)) \end{aligned}$$

よく見ると f の数が自然数に対応していることがわかるのではないであろうか．結局，1 を f 一つ，2 を f 二つ，3 を f 三つ，で示しているのがチャーチ数である．これが数の定義である．これは Figure 3 に示すように Figure 1 の表現とほぼ同じことがおわかりだろう．

マービン「長かったですねえ．ブログ 9 回目でやっと数を定義できただけですか．Wikipedia の Page ではチャーチ数が出てきた時には λ の定義も終ってますよ．しかも単なる「四角の数」で数を定義するという単純さ．いや単純さが悪いわけではないのですが，簡単なことを学者ぶって単に難しく言っているだけに見えますねぇ．どうやって計算するのかもまだこれからとは．私のように宇宙を 5 回も経験した者でもこんなに遅いと気が滅入りますね．」

しかし，マービン，チャーチ数がどんなものかはわかってもらえたのではないだろうか．チャーチ数をそのまま出しても Peano の定理がないとこれが自然数だとはわかりにくいのではと考えたのだ．Peano の定理それ自体も定義だけ読んでも味気ないと思うのだが．

マービン「どうですかねえ．私は疲れました．まあ数は定義できたのかもしれませんが，計算に不向きと言われているローマ数字の方がまだましのような．2008 を書くのも大変だ．私のコプロセッサも気が滅入ったようです．」

ようやく数が定義できたので，次は計算について考えよう．

Figure 4. SUCC mark I (Sirius Cybernetics corp.)

Figure 4: SUCC Mark I (Sirius Cybernetics corporation)

8 手順を実行する機械 SUCC mark 1

パスカルは親父さんの税金の計算の手伝いのために計算機を作ったそうである．大量の計算を手で行うのは大変である．私も計算が苦手であり，計算機さえあれば計算を自分でしなくて良いので大学で計算機を学んだ．ところで時々計算機の専門家ならば計算は得意であると思っている人がいるので困る．暗算が得意ならば計算機なぞ不要であり，学ぶ必要はないのだ．空が飛べれば飛行機など作らなくても良いだろうし，遠くの人と心を瞬時に通わせることができれば，電話など発明する必要もない．私は計算が苦手である．だから計算機科学を学んだのだ．楽をするために計算機を作ってみたいと思ったのだ．

Figure 4 はシリウスサイバネスティック社製の SUCC 1 号機という計算機である．この計算機は一つのチャーチ数を入力としてその次の数を生成する計算機である．しかしこの計算機はもちろん人間のように数の概念を持っているわけではない．入力が与えられると後に示す手順を繰り返すだけである．ある意味自動販売機と同じである．自動販売機も数を理解しているわけではないが，お釣りの計算ができるように，この計算機も計算ができる．

SUCC mark I の計算の手順は次の通りである．

1. 入力が in の上に与えられる．(Figure 5．初期状態．入力にチャーチ数 0 が与えられた場合．チャーチ数 0 は丸一つに四角一つで示されたことを覚えているだろうか．)

2. SUCC 1 号機は左から順に同じ形のタイルを out に並べる．(Figure 6．入力のコピー)

3. SUCC 1 号機は出力の最後に一つだけ四角のタイルを置く．(Figure 7 計算終了．)

この図では，チャーチ数 0 を入力したら，なんとチャーチ数 1 が出力された．チャーチ数 1 を入力したら，チャーチ数 2 が出力されること

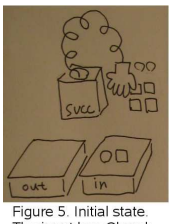

Figure 5. Initial state. The input has Church number 1.

Figure 5: SUCC Mark I. Initial state

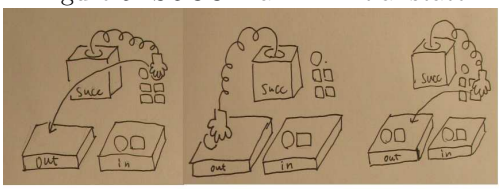

Figure 6. Scan the input and replicate the 'symbols' (circle and square) on the output

Figure 6: SUCC Mark I. Scan the input and replicate the 'symbols' (circle and square) on the output

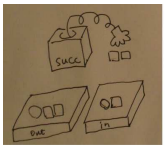

Figure 7. The calculation result of SUCC mark I

Figure 7: SUCC Mark I. The result

はおわかりだろう．ついに計算機ができた! マービンは黙っていて欲しい．マービン「...」

こんな単純なものはばかげていると思うかもしれない．しかし，これは現存する計算機の基礎である．手順を与えることでもっと複雑な計算も可能である．ちなみに SUCC は Successor (次のもの) であり，次の数を生む函数のことである．これがあればどんな自然数も作ることができる．次回はもう少し複雑な計算をする機械を考えてみよう．

9 手順を実行する機械 Pop1

9.1 Pop1 の命令

前回，次の数を生み出す計算機を考えてみた．ここで示された手順は「数を一つ増やす」という手順であるが，手順に意味をつけているのは人間であって．機械は意味を理解することなしに何かを出力することができる.「意味」とは,「理解」とは何かということを考えるのはここでは難しすぎるし，意味のあるような複雑な機構も単純な機構の組合せによって作成可能であるという仮説もある．

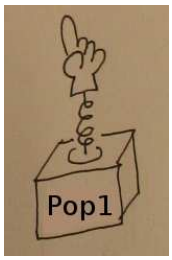

Figure 8: Pop1 (Sirius Cybernetics corporation)

(参照 Marvin Minsky，心の社会) が，SUCC1 ほど単純な機械には知能はないとしてもさしつかえないだろう．ところで，Marvin Minsky 先生とロボットの Marvin に関係があるのかどうかはわからない．これから，意味があるような行動をとれる機械であれば意味が理解されていると考える考え方もある．もちろん通常は人間に意味のあるものの方が望まれるのであるが，機械にはそんなことは関係がない．手順を実行することが計算になったのである．計算とは何かを理解することは計算すること自体には必要がないのである．

まあこれが発達した機械の問題であることはよく SF に書かれる通りである．機械は単に命令を実行するだけであり，何かを理解しているわけではないので命令がなくなるまで手順を実行することになる．モラルがない人間であれば殺人機械を作ることもあるだろう (Cordwainer Smith の The Instrumentality of Mankind(人類補完機構) から Terminator まで)．どういう機械を作るかの決定にはモラルが必要である．Philip K. Dick の Variant 2 などは私の好きな例であるが，モラルの話はここで論じるには大きすぎるテーマなので，λ 計算の話に戻ろう．

Figure 8の機械は Pop1 (P-operation 1) である．Pop1 は 3 つの命令を実行することができる．

1. 入力コピー: 入力を出力テーブルにコピーする (Figure 9)
2. 頭尾消去: コピーされた数字の頭 (Head) と尾 (Tail) を取り除く (Figure 10)
3. 頭尾追加: 記号全体に頭 (Head) と尾 (Tail) を加える (Figure 11)

これらはチャーチ数が入力であれば常に実行できる．チャーチ数でない場合，例えば，頭も尾もない場合に命令 (b) の頭尾消去を実行するとエラーになるとしよう．エラーの場合には Pop1 は怒って記号を投げつけてくるので，プログラムを実行する前にはヘルメットの着用することがサイバネティクス社のマニュアルには書かれている．

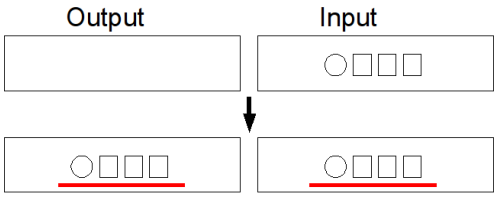

Figure 9: Pop1 procedure (a): copy input to output

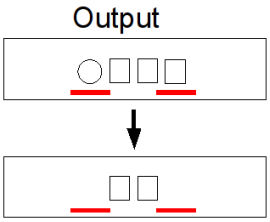

Figure 10: Pop1 procedure (b): Delete head and tail on the output

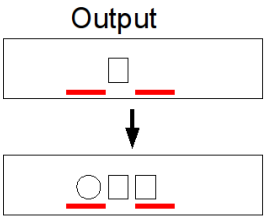

Figure 11: Pop1 procedure (c): Add head and tail on the output

次回はこの命令を使って Pop1 のプログラムを書いてみよう．

9.2 Pop1 Program

前回は Pop1 が実行できる命令を示した．Pop1 のプログラムは実は固定されていて，変更することができない．サイバネティクス社はプログラムを最重要機密として，社内の人間でも自由に見ることはできない．実はプログラムを開発しているプログラマーも開発しているプログラムを見ることができない．しかし，プログラムを実行させてみると Pop1 がどの命令を実行しているのか教えてくれるので，それを書いてみたのが Figure 12 である．ここで (a), (b), (c) はそれぞれ Figure 9, 10, 11 である．Pop1 には実は命令は 3 つしかないことに気がついただろうか．

マービン「こんな計算機に 3 つも命令があるとは無駄なことを.」もちろんいかなる計算機も一つの命令があれば，現存するいかなる計算機ともチューリング等価であることは知られているが，人間には理解というものが必要ということをマービンは忘れている．(一命令計算機にはいくつ

1. (a) 入力コピー
2. (b) 頭尾消去
3. (a) 入力コピー
4. (b) 頭尾消去
5. (c) 頭尾追加

Figure 12: Pop1 program

かの種類はあるが，私の好きな例は，Computer Architecture: A Quantitative Approach という本にある例である．)

プログラムを書いたことがない方はこれがプログラム? と思うかもしれないが，これがそうである．結婚式のプログラムでもオリンピックのプログラムでもテレビのプログラムでもある種の小さな手続きの集りであるように，Pop1 プログラムもこのようにできている．

実行の手順を Figure 13 に示す．

さて，このプログラムはいったい何をするのか? 別に「人間にとって」意味などない手続きでも良かったのであるが，ここではある計算を考えた．Pop は 2 つの入力 A, B をとり，一つの出力 C を返す．ここで A, B, C は変数である．

- Pop1(A, B) -> C

ここで A を 1, B を 2 とすると，

- Pop1(1, 2) -> 3

となる．これは数字 1, 2 をとり，3 を返している．実は，次のような関係がある．

- Pop1(0, 0) -> 0
- Pop1(0, 1) -> 1
- Pop1(0, 2) -> 2
- ...
- Pop1(1, 0) -> 1
- Pop1(1, 1) -> 2
- Pop1(1, 2) -> 3
- ...
- Pop1(2, 0) -> 2
- Pop1(2, 1) -> 3
- Pop1(2, 2) -> 4
- ...

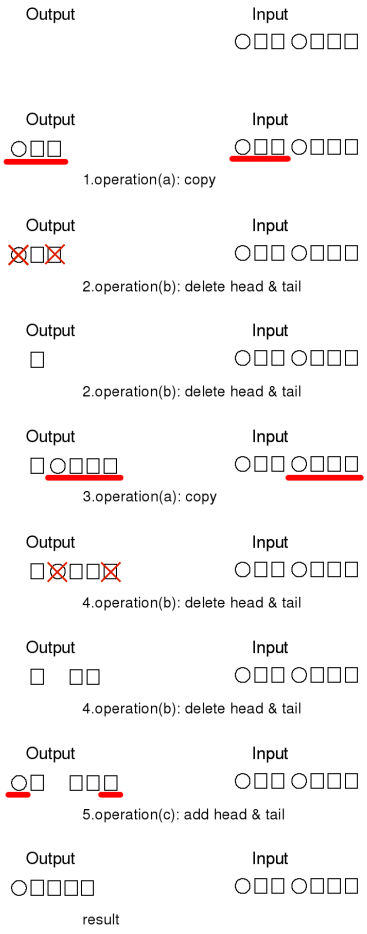

Figure 13: Pop1 program illustrated

Pop1 の正体がわかっただろうか? これは加算 (Plus operation) である．ついでに何故 Pop という名前なのかもわかっただろう．もちろん名前は人間のためであって，Lambda1 でも Machine1 でも Hokuspokus1 でもかまわない．これは個人的な趣味である．(佐藤・桜井の本に習った)

見慣れた記法に書き直してみよう．

- Pop1(0, 0) -> 0: $0 + 0 = 0$
- Pop1(0, 1) -> 1: $0 + 1 = 1$
- Pop1(0, 2) -> 2: $0 + 2 = 2$
- ...
- Pop1(1, 0) -> 1: $1 + 0 = 1$
- Pop1(1, 1) -> 2: $1 + 1 = 2$
- Pop1(1, 2) -> 3: $1 + 2 = 3$
- ...
- Pop1(2, 0) -> 2: $2 + 0 = 2$
- Pop1(2, 1) -> 3: $2 + 1 = 3$
- Pop1(2, 2) -> 4: $2 + 2 = 4$
- ...

ついに計算機ができた．計算機とは計算をする機械のことである．昔はコンピュータという職業があり，多くは女性だったそうである．この計算機は入力を間違えると怒るほど賢いくせに，数が何かということは人間のようにはわかっていない．単に教えた手順を実行するだけである．しかし，結果として計算しているように見える．この「見える」というのが重要である．現在の計算機も人間のように数を理解しているわけではない．ある手順を機械的に実行すると，それが計算したように見えるだけである．

見掛けと内容，名前と意味はそれぞれ異なるものである．しかし，formarization というのは見掛けが同じならば内容も同じとする考えである．浅い考えと言うなかれ．完全なコピーであれば，それは同じものと考えることができるのである．例えば，ソフトウェアはコピーであるがその機能はまったく同じである．これは工業的には重要なことだ．もちろん，人間の感情とかいうものまで完全にコピーするのは気持が悪いが，私は二台のコンピュータや，映画の DVDdisk には個性は求めない．それらを違う店で買っても同じものが入手できることを期待する．違っていたら逆に嫌である．

一方で友人から手紙をもらう時には直筆のものの方が嬉しかったりする．数学の中に永遠に不変の真実を求める一方で限りある命を愛しく思うことは一見矛盾しているように見えるかもしれない．宇宙を 5 回も見てきたマービンは多分違った意見を持っていることだろう．

9.3 一つ上の考え

これまで SUCC1 や Pop1 という機械を見てきた．それぞれの機械はある手順を実行する．手順は入力に対してどんな操作をして出力を作るかという命令を順番に並べたものである．ここでは単純なモデルとして機械は一度に一つの命令しか実行できないものを考えた (つまりここでは並列計算は考えない)．命令には,「入力があれば出力のテーブルにコピーする」や,「出力のテーブルの頭と尾の記号を削除する」というようなものがあった．

命令を並べた手順によって計算というものができることはこれまでにみてきた．その具体例は SUCC1 と Pop1 であった．こうやって任意の機能の機械を作っていくこともできる．しかし，毎

回新しい問題ごとに設計しなくてはならないのだろうか．足し算のために足し算機械を，引き算のために引き算機械を作る必要があるのだろうか．それとも足し算も引き算もかけ算も割り算もできる，より一般的な計算機械を作ることはできるのだろうか．

こうなると数学者は考える．「いったいいくつの命令があれば全ての問題を解くことができるのだろうか．」「全ての問題とはいったいなんだろうか」「何が計算でき，何が計算できないものなのか」

以前，λ 計算の動機という話をした．そこでは計算というものを形式的に考え直すという動機があった．これまでに計算はより単純な手順によって構成されることを見てきた．つまり，計算は形式的なものでも実行できる，それは可能だったのだ．可能性がわかったので，質問は次のレベル，「ではどうやるのか」，になる．多くの学問の発展と同じパターンがここには見られる．

λ 計算について次のレベルに行く時が来たようだ．

10 抽象化 — 無限は有限の中に存在できる

ここで抽象化ということを述べておこう．抽象化というのはある考えから「なにか特定の具体例とは関係ない考え」を導くものである．この blog は正確さよりも理解したというような感覚を重要視しているので，ここで「なぜこんな『抽象化』などということを考えるのかという動機」を述べておかねばなるまい．抽象化をするのは，具体例では十分ではないからだ．足し算とは何かということがもし理解できれば，「無限の組み合わせの足し算」が可能である．記憶には限りがあるため，個々の例をいくら覚えても有限の組の足し算しかできない．「どんな数でも足し算はできるし，その方法もある．」ということを理解できることは強力である．

機械に問題の具体例を教えこむことによって計算しているように見せる方法もある．これは計算の手順を教える方法とは異なるアプローチである．1+1 の答えは 2 である．1+2 の答えは 3 である．ということを覚えこませると，1+2 は? と尋ねると 3 と答えるので計算しているように見える．しかし，覚えていない答えを答えることはできない．具体例では十分ではないというのはそういう意味である．

1, 2, 3 という具体的な数字ではなく，それらを全てまとめて「数」という概念に抽象化することができる．抽象化によって「無限の種類の問いに答える」計算ができる機械を作ることができる．ここでの無限というのは，どんな数でも足し算できるということであって，普通の人でもできることだ．1 + 10 はできるが，1 + 128 はできないというようなことはない．時々「無限」という言葉がでるとどんなものでも不可能と考える人がいるが，そんなことはない．何か一つ数を選ぶということは無限の可能性の中から一つ選ぶということだ，そしてその数が何であれ，1 を加えることはできるだろう．もちろん数が大きすぎて計算することは一生かかってもできないかもしれないが，しかし，どんな数でも，具体的な数が示されればそれらを足し算することは可能であることは理解できるだろう．

高校数学で一番重要な概念は微分積分だと私は個人的に思っている．それは，収束という考えを通してでてくる，「無限は有限の中に存在可能である」という考えである．たとえば，0 と 1 の間には無限の数が含まれている．無限平面を一点を除いて半径 1 の半球に投影することができる．など例は多数ある．ピタゴラスは有限の区間には有限の数しか入らないとしたため，これに反対したゼノンはパラドックスを唱えて間違いを指摘した．

個々の数ではなく数を抽象化して，数そのものを考えることで無限を扱うことができる．これは実に強力な考えである．λ 計算では計算するということを 3 つの定義に抽象化する．3 つの定義はそれぞれ函数を対象とする．これは具体的な函数ではなく，任意の函数である．つまり無限の種類の函数を考えることができる．

1, 2, 3, 4, ... という具体的な数を，「数」という名前で抽象化し，それ (数) が，全ての具体的な数を含むことが理解できるのは人間のすばらしい能力である．私にとって面白いことに話すことができる能力とこの数を数える能力はどの人でも病気などでない限り可能なようだ．機械には (まだ) それはできない．

人間はこの「抽象化」ということを難なくやってのける．たとえばほとんどの人は椅子という概念を持っている．機械で椅子の写真を探させるのは現在では熱いトピックである．人間は IKEA (2009 年時点で有名な家具の会社) のカタログを見て，いろんな形の椅子を椅子と認識できる．そしてどのように使えば良いのかも簡単に判断できる．しかし，椅子という言葉には，無限の可能性の椅子が含まれているのだ．無限の可能性にもかかわらず人間は新しいそれまで存在していなかったデザインの椅子を認識できる．

同様に，λ 計算では個々の函数を扱わない．函数全てについて考える．どのような数学的問題が未来に出てくるのかはわからない．しかし，あらゆる函数に関しての理論があれば，未来になっても我々はそれを解くことができるだろう．ただし函数という概念が十分抽象化されていて多くの問題をカバーできる場合に限る．数学は古くならな

い．足し算についての考えは，少なくとも何千年かは生きのびてきた．おそらく，かなり長い間この考えは正しいものであるだろう．λ計算が無限の種類の函数について考えたり，抽象化などということをするのは，同様のことを期待するからである．私は一度正しかったものは，いつまでたっても正しいような学問が欲しい．なぜなら一度勉強すれば済むからである．多くの学問はそれを目指しているが，実際はなかなか難しい．

数学が無限とか定義とかに拘るのは，今日正しかった足し算が明日間違いにならないことを期待したいからである．数学によっていろいろな技術が発展してきたのを見ると，私にはこれがとても面白い．一度確立したら無限に変化しない学問が，世界を変化させているからである．しかし，これは当然である．技術の進歩は昨日の進歩の上に成り立っている．したがって，その基礎が毎日変化しては困る．技術自体は毎日変化してもかまわない．その基礎がゆるぎのないものでさえあれば良い．基礎がゆるがないものであるから，その上に変化を積み重ねられるのだ．

11 λ計算，函数，名前

11.1 函数 = λ

λ計算で最初に私がとまどったのは函数がλしかないことであった．まあ，本当は函数をλと呼ぶためにλしかないのだ．λとは函数のことであるから，函数を「函数」と呼ぶのは当然なのだが，慣れないとどうも納得できないものだった．λ計算以前の函数の概念と多少違う函数なので，そのまま函数と呼べば混乱を招くかもしれず，区別のために何か他の名前が必要だったのだろう．

とにかく函数というのはλである．何故記号「λ」が函数を表現するものとして選ばれたかは諸説あるが，ここでは立ち入らない．(logic で使われたから L のギリシャ文字λを使ったとか話はあるが，著者が知らないことなので「立ち入らない」と偉そうに書いてお茶をにごしておくことにする．)

次に謎だと思ったのは calculus である．数学だとこういうものは Algebra ではないのかなと思っていた．記号論理学 (symbolic logic) ではこういうものも calculus と言うようだ．しかし，私は所詮素人なのでよくわかっていない．

11.2 λと名前

数学では函数の名前として f とか g などを使っていた．その理由は函数が2つ以上あったときにどちらかを区別することができるからである．また英語という言語では函数を function と言うので函数の名前として f を使うようだ．しかし考えてみるとよほど特殊な函数でない限り，どんな函数でも $f := ...$ と書いている．$f(x) := x$ だったり，$f(x) := x^2$ だったり，$f(x) := \sin(x)$ だったりする．

函数が違っても常に $f := ...$ と書くのであれば，別に f にする必要はない．重要なのは ... の部分であって，f は函数についた荷札のようなものである．荷物を受けとることができれば，荷札は別になんでもかまわない．

政府というものを持っている星にはどこにでも役所があるし，貨幣というものを持っている所では銀行というものがある．そういう所では法律によって人はある程度以上待たなくてはならないことに決まっている．人を待たせずに仕事をしては逮捕されてしまうのだ．あなたの星でもそうかもしれない．多くの星では順番待ちの際に番号を配布するが，函数の名前はそういう番号札程度の意味しかない．

私は名前で呼ばれたいが，順番が早くまわってくるのであれば，「次，何番の方どうぞ」でもかまわない．「何番」が何を指しているかということが本質的である．たとえば 10 番が私を指しているのであれば，私は 10 番でいいわけだ．「10 番さん」どうぞ，でいっこうにかまわない．別に私は 10 番が好きなわけではないし，番号札が 10 番でなければ受けとらないというわけでもない．だから，「番号」と「私」の間の対応が私には重要であって，番号は 10 番とか 42 番でも 156 番でもかまわない．

名前は二の次であるということを繰り返してきたが，名前と実体間の対応は重要である．名前は対応のためにあると言っても良いだろう．あるものを一まとめにして名前をつけてそれを呼ぶというのは偉大な抽象化の一歩である．

しかしあえて名前を避けることによって本体を明らかにしようというのがλの目的である．無機質で名前のように思えない「λ」を「函数」の意味で使うのはそういうわけである．一方で理解のためには名前ほど重要なものはないのだが，こんなことを言うと..

マービン「矛盾だ．名前はどうでも良いのに重要とは．ああ，気が滅入る．」

対応こそが重要であり，それは名前で決まるので「人間には」名前が重要になるのだが，名前そのものは 10 番とかのような「人間にわかりにくい」ものでもかまわない．わかりにくいだろうか．

コンピュータは，英語ではコンピュータ，日本語では計算機，ドイツ語では Rechner という名前である．しかし，実際のハードウェアをどう呼ぶかとそのハードウェアの名前という違いがある．1 が英語では one, ドイツ語では Eins，日本語では ichi と呼ばれているが，しかし，それは

1 なのである.「薔薇はどの名で呼ばれようとも甘く香る.」のように，1 を呼ぶ呼びかたは幾通りもあるが，1 という概念そのものの方が本質である．しかし，人間は名前を通じてしか理解しないのでその意味では名前が重要なのである．これを哲学的問題と思う人もいるようだが，そんなに難しいことではない．scheme のような計算機言語では lambda とその名前を変数に bind するという機械的な操作を行う．これは機械に可能な手順であって，人生とは何かというような問題のようには難しくないと思う．もちろん良い名前を考えるというのは十分哲学的な難しさがあるとは思う．(ここでは哲学 = 難しいもの，というステレオタイプを用いている)

12 λ 計算の初歩

函数も結局名前はどうでも良いのであって，そのものが何かが重要である．そういうわけで，函数自身は，函数であることさえわかれば良い．λ は函数ということを示すだけである．ただ，名前は重要視されていないので，それが何かということを直接書くのが重要だ．名前をつけたければあとで適当に名前との対応をつければ良い．(これを bind するとか束縛するとか言う．)

以前，函数の説明で自動販売機のアナロジを使ったのは，函数は何かを入れると何かが出てくるものであるからだ．自動販売機はお金を入れると商品が出てくる．最も簡単な函数は，入れたものがそのまま出てくるものだろう．そんな自動販売機ならば私にも作れそうだ．そういう函数は「何か」を入れたら「何かそのもの」が出てくるものだ．○を入れたら○が出てくるわけである．そういう函数を「λ ○. ○」と書こう．ここで最初の○は入力を，後の○は出力を示すとしよう．ここで○は「何か」を表す.「何か」とは何か? と聞きたくなるかもしれないが,「何か」を具体的なものにするとこのシステムはあまり有用でなくなる．抽象化の一部分である．

たとえれば，ここでの自動販売機はどんなものでも売ることのできるすごいものである．もしジュースしか売れないのであればたいしたものではない．だから「何か」を売る自動販売機なのだ．

なんでこう書くのかというのをちょっと考えてみよう．いきなりこう書こうと言われても，数学者じゃなければ「何で?」と思うのは普通である．数学者はこういうのはゲームのルールと同じであることを知っているので,「これがルールだ」と言うだけで納得してくれる．とはいっても一応数学者なりに便利なルールを作っているのが普通であるのでちょっと説明してみよう．

まず，最初の「λ」はこれが函数であることを示す記号である．函数だとわかればなんでもいいので,「これから私は函数を書くぞ」と書いても良い．その場合には,「これから私は函数を書くぞ ○. ○」となる.「.」より前の「○」は入力を示し，最後の○は出力を示す．これも定義だから他の方法でも良い．例えば逆に書いても良い．あるいは入力と出力を明確にするために,「入力は」と「出力は」という言葉を補っても良い．その場合,「これから私は函数を書くぞ入力は○. 出力は○」となる．

「これから私は函数を書くぞ」と毎回書くのは面倒なので．それを λ と書くと,「λ 入力は○. 出力は○」となる．ここで面倒だから λ と書くというのは冗談ではなく，極めて真面目である.「入力は」「出力は」というのも順番を決めておけば必要ない．結局ルールさえ決めておけば「λ ○. ○」で十分である．ただ何で λ なのかは正直言って私も知らない．でもそれは何故函数は函数と書くのですかという質問と同じである．あるいは 1 を 1 と書くのは何故かとか，滝はなぜ滝というのかというのを考えるのは面白いだろうが，それでは話が進まないのでここで打ち切ろう．ところで，マーク・トゥウェインさんは滝についてはアイデアがあるようだ．

もし，○を入れたら△が出てくるのであれば,「λ ○. △」であるのだが，これでは○と△の関係がわからない.「λ ○. ○」の関係はわかり易い．なぜならば同じということがわかるからだ．自動販売機のアナロジを思い出して欲しい.「λ ○. ○」では，10 ユーロ入れたら，10 ユーロそのままが出力される．5 ユーロを入れたら 5 ユーロがそのまま出力される．

「λ ○. △」の場合，□を入れたら何が出てくるのだろうか．この場合，λ 計算ではやっぱり△が出てくるのである．つまり「λ ○. △」と「λ □. △」は同じ意味であることに注意が必要である．もっと言えば,「λ ○. △」は何を入れようが△を出す自動販売機である.「λ ○. ○」とは，同じものが出てくるという意味であって，実際に○を入れるわけではないのだ．うーむ難しいなあ．変数を御存知の場合には，○は変数であると言えば良いのだが，それでは不親切であろうから次でもう少し説明してみる．

もう一度この入力と出力に使われている○について話をしよう．○というのは「何か」を示している．

マービン: また「何か」... 疲れた．

「何か」って何だ? と言う意見はもっともである.「何か」とはたとえば「お金」を示しているとしよう.「お金」を入れたら「(同じ) お金」が出てくるというのが,「λ ○. ○」である．増えたりも減ったりもしない．入れた分だけ出てくるのである．だから，100 アルタイルドルならば 100 アルタイルドル出てくるわけである．200 アルタイルドルならば 200 アルタイルドル．(何らかの)

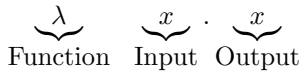

Figure 14: a Lambda expression

アルタイルドルならば (正確に同じ何らかの) アルタイルドルが出てくる.「何か」というのはこういう意味での「何か」である．具体例を言ってしまっては，表現が制限されるので,「何か」というのだ．これは抽象化された考えである．学校ではそういうものを総称して X とする．というふうに習っているので，X と言った方がいいかもしれない．X を入れると X 出てくる．

それならば,「λ □. □」も同じ意味である．□を入れたら□が出てくる．同様に「λx.x」「λy.y」もまったく同じ函数を示している．ただ，λ 計算の人達は慣習的に「λx.x」と書くことが多い.「何か」を入れたら「何か」が出てくるというのはそういう意味である．ここで「何か」って何だ? と聞かれても「だからそれは『何か』です」と数学者は答えるだろう．これはあなたをからかっているのではない．図 14に λ 式の構造を示しておこう．

自動販売機のたとえに戻ろう．100 アルタイルドルを入れた時だけ，100 アルタイルドル分のシリウス行きの切符を売る自動販売機は簡単である．これは行き先も値段も固定されている自動販売機である．このような機械で古いものとしてはアレキサンドラのヘロンの記述がある．

しかし，もし 150 アルタイルドルを入れたら，150 アルタイルドル分のオリオン行きの切符を売ることもできる自動販売機だともっと便利である．そして，クレジットカードを入れたら，宇宙の果てのレストラン行き予約のチケットが出たり，あるいは Kreuzberg のコンサートまで手配できればもっと便利である．つまり「何か」の範囲が広い方が便利だということは賛成してもらえるだろうか．ここでは最初入力の「何か」は 100 アルタイルドルだった，それがどんな値段でも良いアルタイルドルになり，クレジットカードになった．出力の「何か」はシリウス行きのチケットだけだったが，オリオン行き，レストラン，コンサートと広がった．そういう「何か」を本当に「何か」全てとして考えようというのが，函数という考えなのである．

「何かって何だ?」「何かです.」の意味がわかってもらえたらこの節の目的は達成された．最後にマービンのコメントを聞こう．

M「私を設計した技術者は，一般両替機の開発をしていました．普通の両替機と違うのは入力に物を入れるとそれと同等の価値の何かを出すところです．あなたが 3ヶ月かかって書いたソフトウェアを入れたら冷や飯とタクアンが出てきたやつです.」

I:「あの機械 (函数) は壊れていたと聞いているが」

M:「確かに，私はタクアンが付くのはおかしいと思っていました.」

I:「...」

13 λ 計算: 二倍する函数

λ 計算の続きである．何らか入力に対してその二倍を出力するという函数を λ 式で書いてみる．

$$\lambda x.2x \tag{1}$$

前回の図 14 を見るとこの函数は入力が x で出力は $2x$ であるので，これは入力を二倍する函数である．

しかし本当はこれは正確ではない．ここにはごまかしがある．函数とは何かを考えるのに，既にかけ算という函数は何なのか知っている．何を考える時にはもっと基本となる仮定から始めなくてはいけない．それにここでは数まで何かわかっていることになっているが，λ 計算ではこのような数は何かについては規定していないので使ってはいけないのである．機械に計算をさせたいというのが λ 計算を学ぶ私の動機の一つであった．人間には易しいことであっても，機械には難しいことは多数ある．たとえばかけ算などはまだ定義されていない．

しかし，我々には既にチャーチ数がある．後でチャーチ数を使う方法に戻ることにして，まずは函数についてもう少し話をすすめてみよう．

この函数 ($\lambda x.2x$) は，私が習った方法で書けば，$f(x) := 2x$ であるが，$g(x) := 2x$ でも良いわけで，ここでの f と g とかいう名前は役所の順番待ちの番号札の番号みたいなものである．(ここで := というふうに '=' にコロン':' がついているのは定義するという意味である) 名前は通常非常に重要なものであるけれども，それが本質かというとそうではないこともある．λ 式の書き方では，λ を函数と考えれば，$\lambda x.2x$ は "x を引数にとる函数で，それは $2x$" というふうに読んでも良いだろう．こうすると f とか g という名前を書く必要がない．

函数を λ と呼ぶのには多分名前が本質ではないことも関係している．函数の正体が重要なので，別に a でも b でも λ でも良いことを示したいのだ．函数とは何かということを真面目に考える人達がいて，その人達は「名前は何でもいいのだ，本体がいったい何なのかをもっと研究したい.」と思ったのだろうと思う．

14　λ計算: 函数の適用

λ計算でもう一つ重要なのは函数は引数を一つしかとらないことである．引数というのは函数の入力のことである．二つ以上の入力がある場合の話はすぐ後にしよう．

引数の具体的な値が決まったら，つまり入力が決まったら，それを右に書いて函数の「適用」ということをする．入力が決まるというのは，自動販売機で言えば，お金を入れた状態である．自動販売機はお金が入ってくるのを待っており，ひとたび入力が決まれば，出力を計算する．函数も多くの場合，それ自体では計算はせず，何か入力があって計算を開始する．

「適用」というとおおげさな気もするが，とりあえずは函数に値を代入すると考えておいてもいいだろう．あとで値でないものを代入する場合があったり，値をどうこうするというよりも，函数そのものがどういう役割なのかということを考える方が面白くなってくるので，おいおい「函数を適用する」という言い方の方がしっくりくるようになるだろう．

具体的な函数の適用の例を見てみよう．

$$f(x) := 2x$$
$$\lambda x.2x$$

これらは同じ函数である．最初のものは通常の書き方，二番目のものはλ式である．これに 3 を代入する，あるいは 3 に適用する場合を書いてみると次のようになる．

$$\begin{aligned} f(3) &:= 2x \\ &= 2*3 \\ &= 6 \\ (\lambda x.2x)3 &= (\lambda 3.23) \\ &= 2*3 \\ &= 6 \end{aligned}$$

同じ結果になった．両方とも入力を二倍する函数なので 3 を入れれば 6 になる．

λ計算では函数はいつも一つの値しかとらない．では $f(x,y) := x - y$ のようなものはどうするかと言うと，一つの値をとって一つの値をとる函数を返すようにする．

$$\lambda y.(\lambda x.x - y)$$

ここで $\lambda x.x - y$ は，x を引数にとり，$x - y$ を出力にする函数である．これを 3 に適用すると，

$$(\lambda x.x - y)3 = 3 - y$$

が返ってくる．それでこれを y の函数とすると，

$$\lambda y.3 - y$$

である．ここで y = 1 にすれば，

$$\begin{aligned} (\lambda y.3 - y)1 &= 3 - 1 \\ &= 2 \end{aligned}$$

となって，3 – 2 が計算できた．一回の適用で一つ引数が決まるので，これを繰り返していけば引数がいくつあっても問題はない．

函数のように値でないものが返ってくるのは多少気持が悪いかもしれないが，これは強力な道具になる．

シリウスサイバネティクス社の自動販売機には，自動販売機を売る自動販売機がある．シリウスサイバネティクス社は「なんでも売る自動販売機 gensym3141! これ一つであなたもコンツェルンの経営者」とか宣伝しているが，マービンに言わせれば「そんなものが不可能なことは地球人にだってわかる．ああ気が滅入る」ということになるらしい．しかし，こういうふうに何かをする機能のものを取り出す機能というものをλ計算では普通に考えることにする．それも入力をとって出力を返すものだから，やっぱり函数であって，λなのである．

15　壊れている販売機

以前，Subether 通信で壊れている自動販売機はどうなるのかという質問があったのでその話をしておこう．自動販売機は函数のアナロジなので壊れている函数というものは何かということを考えることになる．

まず，壊れているとは何かということを考えてみよう．

1. 何を入れても何も出てこない
2. 何を入れても同じものが出てくる．
3. 何か入れると予想できないものが出てくる．

自動販売機がこのような振舞いをすれば，どれも「壊れている」と言えるだろう．ただし，壊れているという言葉はまだあいまいである．このような振舞いを常にする機械を作ることができるのであればそれは「仕様」であるかもしれない．ここでは仕様通りに動かない機械を壊れていると定義しよう．したがって，仕様であるのか壊れているのかは「仕様」を見ないとわからない．λ式は十分な表現力があるので，これらの仕様も書くことができる．

1. 何も出てこない場合: まず，何も出てこないということを定義しよう．もしこれがアルタイルドルの貨幣交換機であれば，何もでてこないとい

うことは 0 アルタイルドル出てくるということである．したがって，x アルタイルドル (x = 0) 出てくるということで，λx.0 とでも書けば良いだろう．同様に何も出てこないということが 0 出てくるということならばどんなものでも書き下せる．

2. 同じものしか出てこない場合: 何を入れても同じものが出てくるのは簡単である．その出てくるものを y としよう．λx.y と書けば，これはどんな x を入れても y しか出てこない自動販売機である．

3. 予想できないものが出てくる場合: 再び言葉の定義になるが，予想できないという意味をもう少し明確にしてみよう．予想するのは誰だろうか．これは人間が予想できないということであろう．しかし，これは人間には難しいだけであって，マービンならば Mersennely に twisted 乱数発生器でも見抜くことができるかもしれない．それならば，マービンに予想してもらって，そういう函数を書けば良いのである．たとえば，疑似乱数発生器を使って，一見でたらめであるが，実際には人間にはわからないだけにするという方法である．しかし，チューリングさんが提案したように放射性元素を計算機にくっつけて，その元素の崩壊する様を観測して乱数発生器とするという手もある．この場合にはまず予想が不可能のはずである．これは今でも最高のハードウェアによる乱数生成器であろう．

16 自動販売機 gensym3141

シリウスサイバネティクスの自動販売機 gensym3141 のすごいところは無限の種類の自動販売機が出せるところである．これ一台あればコーヒーだろうが車だろうがコンピュータであろうが買うことができる．この機械は自動販売機しか出せないが，コーヒーが欲しければコーヒーの自動販売機をまず買って，その自動販売機でコーヒーを買えば良いのだ．

しかし，無限の種類の自動販売機でも出せないものもあるのが残念なことである．無限の種類の自動販売機を出せる自動販売機ならばどんな自動販売機も出せると思っている人はマービンに怒られて下さい．でも，それでは不親切なので簡単な話をしよう．

自動販売機 gensym3141 はキーパットがついていて，1, 2, 3, ... といくらでも数を売ち込むことができる．というのは番号さえ決めれば無限の可能性がある．たとえばボゴン人の現代詩集を売る自動販売機は 157079632679489661923 である．しかし，イルカの作った金魚鉢を売る自動販売機は -111111 という番号がついているのだが，自動販売機 gensym3141 には - の記号がない．したがって，無限の数が入力できるにもかかわらず，買えない自動販売機があるのだ．とりあえず無限とは全部ではないということさえわかってくれれば良い．

もちろんシリウスサイバネティクス社の営業部長は無限の種類が出せるのだから，全部の自動販売機が出せるということを客に納得させるためのトレーニングを積んでいる．そして宇宙は広いので，それに納得してしまう馬鹿な客もいくらでもいるのである．したがって，宇宙は自動販売機 gensim3141 であふれているのである．

ところで，自動販売機 gensym3141 で買えない有名な自動販売機はいくつもある．まず，自動販売機 gensym3141 は自動販売機 gensym3141 で買えない．自動販売機 gensim3141 を出そうと考える，と自動販売機 gensym3141 に内蔵されているシリウスサイバネティクス社製のマインドコントロール装置が作動して記憶喪失になるのでくれぐれも考えない方が良い．ある人がシリウスサイバネティクス社製のマインドコントロール装置を無効にする装置を売る自動販売機を出そうとしたが，シリウス社は当然その場合には本人がシリウス社に忠誠を一生誓うというマインドコントロールにかかるような装置を自動販売機 gensym3141 に内蔵済みである．

これはシリウスサイバネティクス社の保有する大量の特許の一つで自動販売機 gensim 3141 はシリウスサイバネティクス社からしか買えないのである．特許を買う自動販売機は存在するが出せないらしい．詳細は不明である．誰かが犠牲になって試してみる必要がある．

また，ガイド社もガイドの専売権を持っているのでガイドを売る自動販売機も出せない．ガイドの専売権を売る自動販売機の専売権もガイド社が持っている．シリウスサイバネティクス社とガイド社はガイドの専売権を売る自動販売機の専売権を売る自動販売機の専売権がどちらにあるかで長年訴訟で争っているが，これはなかなか決着がつきそうにないのである．それが終われば次の訴訟が起こることは目に見えている．

gensim3141 の話は十分だと思うので，そろそろ λ の話に戻ろう．

17 λ版 チャーチ数 (再掲)

以前，チャーチ数を箱で作った．1, 2, 3, というような数を機械にもわかるようにどうやって定義すれば良いのかという話だった．この時には計算の話をしていたので具体的な数が欲しかったのだ．数なしで計算の話をするのはなかなか難しいので不自然な話の流れだったかもしれないが，著者の力量ではそんなものである．

以前既に紹介したが，チャーチ数の λ 版を再掲しておこう．このあと使う予定だ．

$$\begin{aligned} 0 &:= \lambda f x.x \\ 1 &:= \lambda f x.fx \\ 2 &:= \lambda f x.f(fx) \\ 3 &:= \lambda f x.f(f(fx)) \end{aligned}$$

ここで重要だったのは，f の数が番号に対応していることだ．0 を見てみると，函数の入力は f と x だが，その中身は x しかない．1 は f x で f が一つである．

$$\begin{aligned} 0 &:= \underbrace{\lambda}_{\text{Function}} \quad \underbrace{fx}_{\text{Inputs}} . \underbrace{x}_{\text{Output}} \\ 1 &:= \underbrace{\lambda}_{\text{Function}} \quad \underbrace{fx}_{\text{Inputs}} . \underbrace{fx}_{\text{Output}} \end{aligned}$$

しかし中国ではこのようなものは昔から存在していた．1 を漢字で書くと「一」棒が一本，2 を漢字で書くと「二」棒が二本，そして，3 を漢字で書くと「三」と棒が三本である．以下，4 も 5 も棒の数を増やしていけばこれはチャーチ数そのものである．が，しかし，古代の中国人は知恵があったのでこれ以上のチャーチ数は計算機がでてくるまで実用的でないと判断したようだ．ローマの番号も番号が大きいとここでのチャーチ数のようになる．ところで漢字の 0 は 零である．しかし，いつ 0 が認識されたのかはわからない．日本の古典落語ではこほめで 0 を只と言っていた．つまり値段が「ただ」のただである．

18 λ版 SUCC

18.1 SUCC とは

Peano の定理で重要だったのは Successor 函数であった．覚えているだろうか．次の数を作るという函数である．実際に SUCC1 という機械の手順を既に説明した．

何が数かという意味で数を定義する際，全部の数を並べることで数を定義してもかまわない．しかし数学者は怠けることが仕事であるし，無限のものを並べるのはたいへん難しい．こういう問題に対して数学者は最初の数と「次の数を出すもの」を作って，あとはそれぞれの人に適当にしてもらうことをする．具体的な数 1,2,3, ... をそのまま使うのではなく,「数」というものの性質を抽象化してそれを数の定義にするのだ．抽象化の話は覚えているだろうか?

「次の数」を作るもの，そういうものは函数であるので λ である．最初の数とこのような λ があれば，全部の数が出せるはずだ．自動販売機 gensym3141 は，自動販売機を入れると，次の自動販売機を出す自動販売機 (自動販売機 6931471805) も売っている．自動販売機 1 は「ニシン販売機」であり，自動販売機 2 は「サンドイッチ販売機」であり，自動販売機 3 は「ニシンサンドイッチ販売機」である．自動販売機 6931471805 は自動販売機 1 を入れると自動販売機 2 を出す販売機である．これはまた，自動販売機 2 を入れると自動販売機 3 出す．じゃあ，自動販売機 6931471805 を入れるとどうなるか，もちろん自動販売機 6931471806 を出すに決まっている．これは数ある自動販売機の一種に過ぎないが，次の自動販売機を出す自動販売機は知的生命体には特別なのものとして感じるので，SUCC と呼んだりする．それはこんな形の λ として書くことができる．

$$\text{SUCC} \quad := \quad \lambda\, n\, f\, x.f\, (n\, f\, x)$$

多分 Wikipedia を読んだ人は，この λ 式を適用しようとしただろう．で，簡単に適用できた人はここから先を読む必要はない．私はできなかったので一週間考えてしまった．それで，多分こうだろうというのがわかったので，この文章を書いている．Wikipedia の補足のようなものだが，もし実際に適用してみたい人にはきっと面白いだろう．

18.2 左結合と右結合

λ 式では函数の適用は左結合という規則を忘れていたので追加しておこう．これは

$$fxy = (fx)y$$

と書くが，まあ，左側から処理するということで良いだろう．たとえば，λ 式ではないが，$1-2-3$ は，左結合であり，$(1-2)-3$ となる．(括弧のある方が先に計算される) したがって，

$$\begin{aligned} 1-2-3 &= (1-2)-3 \\ &= -1-3 \\ &= -4 \end{aligned}$$

である．これが右結合であれば，

$$\begin{aligned} 1-2-3 &= 1-(2-3) \\ &= 1-(-1) \\ &= 2 \end{aligned}$$

となって答えが異なる．λ 式で書けば，

$$\lambda x.\lambda y.\lambda z.x-y-z$$

に 1 2 3 を適用すると，

$$
\begin{aligned}
&(\lambda\underline{x}.\lambda y.\lambda z.x-y-z)\ \underline{1}23 && \cdots \quad \text{x に 1 を適用}\\
&=(\lambda\underline{y}.\lambda z.1-y-z)\ \underline{2}3 && \cdots \quad \text{y に 2 を適用}\\
&=(\lambda z.1-2-z)\ 3\\
&=(\lambda\underline{z}.-1-z)\ \underline{3} && \cdots \quad \text{z に 3 を適用}\\
&=-1-3\\
&=-4
\end{aligned}
$$

ということになる．左結合ならば $x=1, y=2, z=3$ であって，右結合ならば $x=3, y=2, z=1$ であり，何も言わなければ左結合となる．

函数の適用は左からというルールを前回述べた．まず，左結合とか右結合とかいうことをどうして考えるのかというのは，同じ式でも処理の方法で答えが違うのは困るからだ．$1-2-3$ を $(1-2)-3\ (=-4)$ と計算する場合と $1-(2-3)\ (=2)$ とする場合では答えが異なる．答えが毎回違うという計算機でも使えるという人もいるかもしれないが，普通はそうでは困るということは賛成してもらえると思う．式が同じならば答えが同じ計算機を使いたい場合には，どう計算するかを定義しておかねばいけない．それが左結合とか右結合とかいうことを考える理由である．

左結合か右結合を決めなくてはいけないのは上記の理由によるが，なぜ左結合なのかとか思った人がいるかもしれない．これも定義で右か左かはどちらかに決まっていればかまわない．左結合にするというのは，誰か人が決めたのである．どっちにするかは最初に決めた人次第だが，慣れのせいだろうか，これが私には自然に感じるので不思議なものである．しかし，自然に感じるというのは不自然な気がする．まあ，定義なので，そのものにはあまり意味がないと思ってもかまわない．右結合の書き方でも定義は可能である．左結合なのは，現存する数学の記法がヨーロッパで発達したものであり，既存のヨーロッパの言語が左から右に書くからであろうと個人的に推測する．

函数を左結合で評価する，というような定義はスポーツのルールのようなものである．バスケットボールのルールではボールをドリブルしなくてはいけないが，なぜそうなのかというのはルールだからである．野球でボールをドリブルしても意味はなく，サッカーで手を使ったら反則である．同じボールを使ったゲームなのに何故かと聞かれてもルールであるから仕方がない．数学にはそういう一面がある．「何故左結合なのか？」「定義だから．(ルールだから)」

この数学のルールを何か不変で唯一な真実というふうにどこかで教えられてしまうと不幸が初まる．$1+1=2$ というものを不変で唯一な真実だと教えられると後で数学でつまずいてしまう．たとえば，$1+1=1$ という数学で多くの計算機は動いている．多くの計算機は二進法であり，0 と 1 しかない，つまり一桁の計算では 2 というものがないからである．ないものを使うことはできない．もう一つの不幸は，これから $1+1=2$ は間違いだと思うことである．野球とサッカーのルールのようにそれぞれのゲームにはそれぞれの真実がある．$1+1=2$ というのは有用なルールであるが，不変で唯一な真実ではない．定義された後，つまりゲームのルールが決まれば，それはそのゲームで唯一不変な真実となる．

ただし，スポーツにしろ数学にしろ関心は似た所にある．「そういうルール/定義があったら面白いゲームになるか?」である．実はそのルールや定義そのものは何かというのはあんまり興味がない．むしろ，そのルールの下でのゲームが面白いかどうかが問題なのだ．

足し算というルールを作ったら，何か面白いものがそのルールから出てくるだろうか? 足し算とかけ算だけで作った世界ではどんなものだろうか? というようなことを考える．蛇足であるが足し算と定数倍のかけ算一回だけの数学は線形代数という．

人について考えると「その人が何をしたか」の方が，その人そのもの (体重，身長，外見，名前など) よりも興味深い場合が多い．数学でもルールそのものが面白いこともあるかもしれないが，「そのルールで何ができるか」が通常面白い．時々私も「なんでそんなルールがあるんだ.」ということがわからずに，数学が嫌いになることがある．しかし，それは多分ちょっとずれた考えなんだろうと思う．でも，これを教えてくれた先生はいなかったなあ．なんでだろうね，マービン?

マービン「そんなあたりまえのことを今更教えるわけがないでしょう.」

わかっている人にはあたりまえなので気がつかないのかもしれない．しかしあたりまえということには用心しないといけないと思う．

SUCC 函数をもう一度見てみよう．

$$\text{SUCC} \quad := \quad \lambda\, n\, f\, x.f\ (n\ f\ x)$$

この書き方は，

$$\text{SUCC} \quad := \quad ((\lambda n.\lambda f).\lambda x).f\ (n\ f\ x).$$

の省略形である．“.” から左は函数とわかっているので，3 つのうち 2 つの λ を省略しても意味が失なわわれることはない．もう少し詳しく言うと，カリー化に関係する．これは Haskell Curry に因んだ名前であるが，どうもカリーが最初に考えたのではないらしい (Moses Schoenfinkel と Gottlob Frege)．カリー化とは，ある函数に対して，最初の変数を固定すると，残りの引数を使う函数を得る方法である．これによって複数の引数を持つ函数を単一の引数を持つ函数の組合せで書くことができる．つまり，単一の引数を持つ函

数だけ考えれば残りはその組合せであるので，複数引数の函数は考えなくて良い．なんでこんなことを考えるのかというのは，単一引数の函数の方が複数引数函数よりも簡単であるからである．同じ効果が得られるのであれば，簡単なもののほうが理解しやすい．数学者の怠け癖はここにも見られる．

18.3 Apply numbers

では，SUCC を具体的に計算してみよう．まず私は数を適用するというので，0 とか 1 を適用しようとして困った．まったくの素人なのでそういうことをする．ここでの数というのは Church 数であるので，0 は $(\lambda\, f\; x.x)$ である．

$$\begin{aligned} \text{SUCC } 0 &:= (\lambda nfx.f(nfx))(\lambda fx.x) \\ &= (\lambda fx.f\underline{(\lambda fx.x)fx}). \end{aligned}$$

ところでこの中の下線部分，

$$((\lambda fx.x)fx)$$

だが，最初に何が f に入っても消えてしまう．1 変数で

$$(\lambda f.3)$$

というような場合，

$$g(x) := 3$$

と同じなので，x が何になってもこの函数は 3 を返す．こういう x を λ に束縛されていない自由な変数と呼ぶ．黄金の心号のマービンみたいなものである．何があっても別に関係がなかったり，ザップホッドには常に忘れられてしまう．しかし，それならばいらないかというとマービンが出てこないヒッチハイクガイドのように，まったく面白くなくなってしまうので必要なのである．自動販売機 gensym3141 ではクレジットカードを入れない限り，どんな入力も無効である．クレジットカードが入力されない限り，全ての変数は束縛されないのだ．そういう機能も λ 函数で書くことができる．

式 $(\lambda fx.x)fx$ では最初に f に f が入るがそれは束縛されていない．この λ 式の '.' のあとには x しかないので，消えてしまう．ここでは 2 つ f があるのでわかりにくいという場合は，$(\lambda gx.x)fx$ でも同じである．g に f が入っても g は束縛されていないので f は消えてしまう．

すると，

$$\begin{aligned} &(\lambda fx.x)fx \\ &(\lambda x.x)x \end{aligned}$$

となるが，x に x を代入すると，これは x になってしまう．どの x がどの x なのかわからない人もいるかもしれないので，λ 式の変数の名前を変えてみると，

$$(\lambda y.y)x$$

となる．それは

$$\begin{aligned} f(x) &:= x \\ f(y) &:= y \end{aligned}$$

が本質的には同じ函数であることと同じである．マービンは日本語ではマービンであり，英語では「Marvin」であるが，何語で呼んでもマービンはマービンである．ことわっておくが，変数の名前を変えても良いというのは $(\lambda x.x)$ を $(\lambda x.y)$ にしても良いということではない．何が何に対応しているかはきちんとしておかねばならない．そこまで規則をやぶってしまってはボゴン人だっておとなしくなってしまいかねない．

結局，

$$(\lambda fx.x)fx$$

は

$$\begin{aligned} (\lambda \underline{f} x.x)\underline{f}x &\quad \cdots \quad \text{f は未束縛のため消える} \\ (\lambda \underline{x}.x)\underline{x} &\quad \cdots \quad \text{x を入れると x が出る} \end{aligned}$$

より

$$x$$

であるから，

$$(\lambda fx.f((\lambda fx.x)fx))$$

は

$$(\lambda fx.f(x))$$

である．() は優先順位を示すがこの場合にはつけてもあまり意味はないので，怠け者の数学者は

$$(\lambda fx.fx)$$

と書くかと思えば，甘い．まだ省略できる．

$$\lambda fx.fx$$

さて，これはチャーチ数で見ると，1 である．つまり SUCC 0 は 1 である．

ついに SUCC 0 の計算ができた!

19 SUCC 1

では調子に乗って SUCC 1 を計算してみよう．

$$\begin{aligned}\text{SUCC 1} &:= (\lambda\underline{n}fx.f(\underline{n}fx))\underline{(\lambda fx.fx)}\\ &= (\lambda fx.f((\lambda\underline{f}x.\underline{f}x)\underline{f}x))\\ &= (\lambda fx.f((\lambda\underline{x}.f\underline{x})\underline{x}))\\ &= (\lambda fx.f(fx))\\ &= \lambda fx.f(fx)\\ &= 2\end{aligned}$$

下線部分が置き替わっていくところである．SUCC 1 は 2 であることがわかった．

もし，$((\lambda fx.fx)fx))$ がわかりにくい場合にはまた変数名を変えておこう．$((\lambda fx.fx)fx))$ は $((\lambda gh.gh)fx))$ と同じである．これは式の本質は束縛変数名を変えても変わらないからである．(Wikipedia などの α-簡約を参照のこと．) たとえば，$f(x) = 2x$ は，$f(g) = 2g$ と本質的には同じである．グラフの形は軸の名前が $\{x, f(x)\}$ になったか，$\{g, f(g)\}$ になったか以外は同じである．

$$\begin{aligned}((\lambda gh.gh)fx) &\quad\cdots\quad f \text{ を } g \text{ に代入}\\ ((\lambda h.fh)x) &\quad\cdots\quad h \text{ を } x \text{ に代入}\\ (fx)&\end{aligned}$$

となる．いかがだろうか．

20 ADD

また，加算は以下のように定義できる．

$$\text{PLUS} := \lambda mnfx.mf(nfx)$$

$1 + 2$ を計算してみよう．1 と 2 はそれぞれ

$$\begin{aligned}1 &:= \lambda fx.fx\\ 2 &:= \lambda fx.f(fx)\end{aligned}$$

であった．

$$\begin{aligned}&(\lambda\underline{m}nfx.\underline{m}f(nfx))\underline{(\lambda fx.fx)}(\lambda fx.f(fx))\\ =\ &(\lambda nfx.(\lambda\underline{f}x.\underline{f}x)f(nfx))\underline{(\lambda fx.f(fx))}\\ =\ &(\lambda nfx.(\lambda\underline{x}.fx)\underline{(nfx)})(\lambda fx.f(fx))\\ =\ &(\lambda\underline{n}fx.f(\underline{n}fx))\underline{(\lambda fx.f(fx))}\\ =\ &(\lambda fx.f((\lambda\underline{f}x.\underline{f}(fx))\underline{f}x))\\ =\ &(\lambda fx.f((\lambda\underline{x}.f(f\underline{x})\underline{x}))\\ =\ &(\lambda fx.f(f(fx)))\\ =\ &\lambda fx.f(f(fx))\\ =\ &3\end{aligned}$$

したがって，$1 + 2 = \text{PLUS } 1\ 2 = 3$ である．魔法のように思うかもしれないが，原理は Pop1 のところで説明したものと同じである．ある意味，f の数が数を示すので，二つの数を継げることで加算をするのだ．

つまり，1 = f で 2 = ff ならば，1 + 2 = f + ff = fff である．他の例を挙げると，3 + 4 = fff + ffff = fffffff = 7 である

21 MULT

乗算は以下の様に定義される．

$$\text{MULT} := \lambda mnf.m(nf)$$

ここで，

$$\begin{aligned}2 &:= \lambda fx.f(fx)\\ 3 &:= \lambda fx.f(f(fx))\end{aligned}$$

より，

$$\begin{aligned}&\text{MULT } 2\ 3\\ :=\ &(\lambda\underline{m}nf.\underline{m}(nf))\underline{(\lambda fx.f(fx))}\\ &(\lambda fx.f(f(fx)))\\ =\ &(\lambda nf.(\lambda\underline{f}x.\underline{f}(\underline{f}x))\underline{(nf)})(\lambda fx.f(f(fx)))\end{aligned}$$

f の数だけ (n f) がコピーされることがわかる．f の数がすなわち数を示していたので，たとえば 3 なら 3 回のコピーがなされる．

$$= (\lambda\underline{n}f.(\lambda x.(\underline{n}f)((\underline{n}f)x)))\underline{(\lambda fx.f(f(fx)))}$$

最初の数の分 (= 2) コピーされた (n f) の n に次の数 (= 3) が入る．

$$\begin{aligned}=\ &(\lambda f.(\lambda x.((\lambda\underline{f}x.\underline{f}(\underline{f}(\underline{f}x)))\underline{f})\\ &(((\lambda\underline{f}x.\underline{f}(\underline{f}(\underline{f}x)))\underline{f})x)))\\ =\ &(\lambda f.(\lambda x.(\lambda\underline{x}.f(f(f\underline{x})))\\ &\underline{(((\lambda x.f(f(fx))))x)})\\ =\ &(\lambda f.(\lambda x.f(f(f(((\lambda\underline{x}.f(f(f\underline{x}))))\underline{x})x))))\\ =\ &(\lambda f.(\lambda\underline{x}.f(f(f(f(f(f\underline{x})))\underline{x}))))\\ =\ &(\lambda f.f(f(f(f(f(fx))))))\\ =\ &6\end{aligned}$$

乗算は他の方法でも定義できる．

$$\text{MULT} := \lambda mn.m(\text{PLUS}n)0$$

Wikipedia にはこのように書いてある．ここでの 0 はチャーチ数の 0 であるが，これは初心者に

は不親切な気がする．私は数字の 0 と思ってしまった．

MULT 2 3

$$
\begin{aligned}
&:= (\lambda \underline{m} n.\underline{m}(\mathrm{PLUS} n)\underline{(\lambda f x.x)}) \\
&\quad (\lambda f x.f(fx))(\lambda f x.f(f(fx))) \\
&= (\lambda n.(\lambda \underline{f} x.\underline{f}(\underline{f} x))\underline{(\mathrm{PLUS} n)} \\
&\quad (\lambda f x.x))(\lambda f x.f(f(fx)))
\end{aligned}
$$

ここでも似たパターンである．(PLUS n) が最初の数分コピーされる．

$$
\begin{aligned}
&= (\lambda \underline{n}.(\lambda x.(\mathrm{PLUS}\underline{n})((\mathrm{PLUS}\underline{n})x)) \\
&\quad \underline{(\lambda f x.x))}(\lambda f x.f(f(fx))) \\
&= (\lambda n.(\mathrm{PLUS} n)((\mathrm{PLUS} n) \\
&\quad (\lambda f x.x)))(\lambda f x.f(f(fx))) \\
&= (\mathrm{PLUS}(\lambda f x.f(f(fx)))) \\
&\quad ((\mathrm{PLUS}(\lambda f x.f(f(fx))))(\lambda f x.x))
\end{aligned}
$$

下線部分，PLUS 3 0 は (= 3 + 0) 3 であるから

$$= (\mathrm{PLUS}(\lambda f x.f(f(fx))))(\lambda f x.f(f(fx)))$$

これは 3 + 3 である．したがって，

$$= 6.$$

となった．

22 PRED

引き算をしたいのだが，これまでに作った Church 数では負の数がない．これは f の数で数を示していたためだ．簡単のためにここでは負の数を考えないことにする．

まず，準備として PRED (predecessor) を考える．これは一つ前の数の意味である．しかしここでは 0 の前は考えないものとする．

PRED := $\lambda n f x.n(\lambda g h.h(gf))(\lambda u.x)(\lambda u.u)$.

PRED 2 を例として計算してみよう．

PRED 2

$$
\begin{aligned}
&:= (\lambda \underline{n} f x.\underline{n}(\lambda g h.h(gf))\underline{(\lambda u.x)} \\
&\quad (\lambda u.u))(\lambda f x.f(fx)) \\
&= (\lambda f x.(\lambda \underline{f} x.\underline{f}(\underline{f} x))\underline{(\lambda g h.h(gf))} \\
&\quad (\lambda u.x)(\lambda u.u)) \\
&= (\lambda f x.(\lambda x.(\lambda g h.h(gf))((\lambda \underline{g} h.h(\underline{g} f))\underline{x})) \\
&\quad (\lambda u.x)(\lambda u.u)) \\
&= (\lambda f \underline{x}.(\lambda x.(\lambda g h.h(gf))(\lambda h.h(xf))) \\
&\quad \underline{(\lambda u.x)}(\lambda u.u)) \\
&= (\lambda f x.(\lambda \underline{g} h.h(\underline{g} f))\underline{(\lambda h.h((\lambda u.x)f))}(\lambda u.u)) \\
&= (\lambda f x.(\lambda h.h((\lambda h.h((\lambda \underline{u}.x)f))\underline{f}))(\lambda u.u))
\end{aligned}
$$

ここで一つ f が消える．f の数がチャーチ数を示していたのであるから，f が一つ消えるというのは「引く 1」ということを行ったことと同じである．

$$
\begin{aligned}
&= (\lambda f x.(\lambda h.h((\lambda \underline{h}.\underline{h} x)\underline{f}))(\lambda u.u)) \\
&= (\lambda f x.(\lambda h.h((fx)))(\lambda u.u)) \\
&= (\lambda f x.((\lambda u.u)((fx)))) \\
&= (\lambda f x.(fx)) \\
&= (\lambda f x.fx) \\
&= 1
\end{aligned}
$$

ちなみに 0 を入れてみるとどうなるだろうか．

PRED0

$$
\begin{aligned}
&:= (\lambda n f x.n(\lambda g h.h(gf)) \\
&\quad (\lambda u.x)(\lambda u.u))(\lambda f x.x) \\
&= (\lambda f x.(\lambda f x.x)(\lambda g h.h(gf)) \\
&\quad (\lambda u.x)(\lambda u.u)) \\
&= (\lambda f x.(\lambda x.x)(\lambda u.x)(\lambda u.u)) \\
&= (\lambda f x.(\lambda u.x)(\lambda u.u)) \\
&= (\lambda f x.x)
\end{aligned}
$$

0 になってしまった．これは良い性質とも言える．なぜなら出力が常にチャーチ数の範囲に収まるからである．演算の結果，定義されていないものが出てくるのはあまり嬉しいことではない．詳しいことは「群論」というような分野で考えられている．たとえて見ると，自動販売機販売機械が自動販売機以外のものを出すのは困る．なぜなら自動販売機販売機械でなくなってしまうからである．

PRED は入力が 0 の時は 0 を返し，それ以外は一つ前の数を返す函数である．

ここで面白いと思うのはチャーチ数そのものがプログラムの一部を成している所である．もち

ろんここでのチャーチ数は函数であるから自然と言えば自然である．

$\lambda n.n(\lambda gh.h(gf))$ は 0 を入れると，

$$\begin{aligned}&\lambda n.n(\lambda gh.h(gf)))(\lambda fx.x)\\ =\ &(\lambda fx.x)(\lambda gh.h(gf))\\ =\ &(\lambda x.x)\end{aligned}$$

0 が出力される．PRED では他の項があるので，これ以上は詮索しないが，0 が出力されると，最初の項が無視されることに注意しよう．これはチャーチ数の 0 $(\lambda x.x)$ では f が bind されていないからである．

23 SUB

それぞれの具体例は実際にやってみたいと思わないとあまり面白くないかもしれない．計算がどのように進むのかは確かめないと式の羅列でしかないからだ．

PRED を使って引き算を定義する．

$$\text{SUB} := \lambda mn.n\text{PRED}m$$

SUB 3 2 (= 3 - 2) を計算してみよう．

$$\begin{aligned}&\text{SUB 3 2}\\ =\ &(\lambda mn.n\text{PRED}m)(\lambda fx.f(f(fx)))\\ &(\lambda fx.f(fx))\\ =\ &(\lambda n.n\text{PRED}(\lambda fx.f(f(fx))))\\ &(\lambda fx.f(fx))\\ =\ &(\lambda fx.f(fx))\text{PRED}(\lambda fx.f(f(fx)))\\ =\ &(\lambda x.\text{PRED}(\text{PRED}x))(\lambda fx.f(f(fx)))\end{aligned}$$

PRED が二番目の引数分 (= 2) コピーされた．これがトリックである．引く数が10 ならば，PRED が 10 個コピーされる．

$$=\ \text{PRED}(\text{PRED}(\lambda fx.f(f(fx))))$$

つまり，これは，PRED (PRED 3) であるので，

$$\begin{aligned}=\ &\text{PRED2}\\ =\ &1\end{aligned}$$

つまり 3 - 2 は 1 である．引き算も可能だった．

24 λのおわりに

先日，広告でこんなものを見た (図 15)．この広告は「投資の専門家に: 一番簡単なことから始めよう」というもので，多分，$1+1=2$ というのが一番簡単だと言うのであろう．私は $1+1$ が 2 であることを機械に教えるにはどうすれば良いかということを 8ヶ月かかって説明してきた．(ところでこの広告は煙草の広告である) 人間ならば，確かにこれは簡単かもしれない．しかし，機械にそれを教えるとなると，1+1 とは何かということを追求する必要がある．1+1 の前に，数とは何かということを考え，それをチャーチ数で表現した．

Figure 15: An advertisement at Mehringdamm U-Bahn station

パラノイアかもしれないが，あたりまえのことというものは，簡単だからあたりまえなのではなく，単にいつも身近にあるからあたりまえなだけだと思う．簡単なことではない．家族や恋人と一緒に時間をすごすことをあたりまえに感じてしまっている人もいるかもしれない．しかし，それは単にそうすることに慣れているからであって，簡単だからではないと思う．時々，あたりまえのことをふりかえって見ることも大切だと思う．

これで λ 計算のためのヒッチハイカーズガイドというテーマのブログはとりあえず幕とする．

Wikipedia には λ 計算でチャーチ数を SUCC や PLUS に適用するとそれが正しいとわかる．と書いてあるが，私には一週間もわからなかった．友人らの助けを借りてやっとわかったので，どうやるのかということを書いておこうと考えて始めたのがこの短編である．

ここでは λ 計算とは何か，どうしてそういうものを考えたのか，そして具体的に簡単なものを適用する方法を述べてきた．この短編が私と同じ所でつまづいている人の助けになれば幸いである．

しかし，λ 計算はまだまだ奥が深い．これでは入門の扉を開いたという程度である．私には combinator がまだ理解できていない．いつかわかったらそれについて書いてみたい．書くことによって学ぶことができるのは良かった．あるいは教えることは学ぶことであるとつくづくおもった．本当に興味のあるものでないと続けるのは難しいということも学んだことであった．

この小文は λ を informal な方法ではあるが，感じをつかむことを重点に置いて書いた．ここでは λ 式を構成的に示すこともしなかったし，α-conversion, β-conversion, η-conversion などの基礎的な変換方法も詳しくは述べなかった．さらに詳しく知りたい人は Wikipedia [?] が良い入門となることと思う．

25 まとめ

$$
\begin{aligned}
0 &:= \lambda fx.x \\
1 &:= \lambda fx.fx \\
2 &:= \lambda fx.f(fx) \\
3 &:= \lambda fx.f(f(fx)) \\
\mathrm{SUCC} &:= \lambda nfx.f(nfx) \\
\mathrm{PLUS} &:= \lambda mnfx.nf(mfx) \\
\mathrm{MULT} &:= \lambda mnf.m(nf) \\
\mathrm{MULT} &:= \lambda mn.m(\mathrm{PLUS}n)0 \\
\mathrm{PRED} &:= \lambda nfx.n(\lambda gh.h(gf))(\lambda u.x)(\lambda u.u) \\
\mathrm{SUB} &:= \lambda mn.n\mathrm{PRED}m
\end{aligned}
$$

謝辞

この blog を書くきっかけをつくってくれた内田さん，計算の具体例を理解する手助けをして下さった Hoedicke さんと Rehders さん，そして今回書くことができなかったが，実装のヒントを与えてくれた前田さんに感謝します．他にも助言や感想を下さった皆さまにお礼を申し上げます．また，この blog を読んで欲しかったが，残念ながら永遠に不可能になってしまった楯岡さんにこの blog を捧げます．