# 目次

# 第0章

# まえがき

### 関数型イカ娘とは!?

Q. 関数型イカ娘って何ですか?
A. いい質問ですね！
Q. ついに最終号ですね
B. じゃあ一緒に見届けようか——λカ娘という、存在の結末を

関数型イカ娘とは、「イカ娘ちゃんは 2 本の手と 10 本の触手で人間どもの 6 倍の速度でコーディングが可能な超絶関数型プログラマー。型ありから型なしまでこよなく愛するが特に Scheme がお気に入り。」という妄想設定でゲソ。それ以上のことは特にないでゲソ。

この本は、コミックマーケット 80 での「簡約! λ カ娘」、コミックマーケット 81 での「簡約!? λ カ娘 (二期)」、さらにコミックマーケット 82 での「簡約!? λ カ娘 (算)」、に続く、さらにさらにコミックマーケット 83 での「簡約!? λ カ娘 4」、に続く、もーさらにさらにコミックマーケット 84 での「簡約!? λ カ娘 Go!」、に続く、そしてコミックマーケット 85 での「簡約!? λ カ娘 Rock!」、コミックマーケット 86 での「簡約 λ カ娘 巻の七」、コミックマーケット 88 での「簡約!? λ カ娘 8」、に続く、コミックマーケット 90 での「簡約!? λ カ娘 9」、に続く、コミックマーケット 92 での「簡約!? λ カ娘 10」、に続く、コミックマーケット 94 での「簡約!? λ カ娘 11」、に続く、十二冊目の関数型イカ娘の本でゲソ。関数型言語で地上を侵略しなイカ!

### この本の構成について

この本は関数型とイカ娘のファンブックでゲソ。各著者が好きなことを書いた感じなので各章は独立して読めるでゲソ。以前の「λ カ娘」本がないと分からないこともないでゲソ。

# 第1章

# そして品質への渇望だけが残された

— @master_q

## 1.1 夢と希望

『春はただそれだけで、なんだか新しく生まれ変わった気分になるじゃなイカ!』
「いやぁまったくだよね。こんなことになるなんて学校の卒業までは考えてもみなかったわ。」

そうなのだった。卒業して、学校では半人前だった回路設計の技能を叩き直すためにメーカーに入社したら、社内に **NetBSD OS を使って準リアルタイム組み込み機器を作る**というクレイジーなプロジェクトがあると聞いて、アマチュアプログラマだった私達の心に火がついてしまったのだった。

NetBSD は通常の Unix 系 OS でしょう？ 仮にスケジューラに細工をしたとしても、どの程度の応答性を保証できるものなのだろうか？ そもそも応答性の保証など Unix 系 OS に可能なのだろうか？ 湧き出る疑問と期待と興奮。

『それにしても、OS 開発の経験はほとんどない我々が即戦力になるのカ？』
「平気平気。わからないことがあれば調べればいいんだよ。だって NetBSD はオープンソースなんだよ？」
『それはそうでゲソ。できれば Debian GNU/Linux だったら良かったのにー、でゲソ。』
「そうだね…… 今となっては Linux の方が BSD より勢いがあると思う。その証拠に Linux の方がユーザも多い。」
『**伽藍とバザール**[*1] によるとユーザが多い方が品質が高いということになるじゃなイカ？』
「でも BSD のソースコードは Linux より整理されていると聞くじゃない？」
『**綺麗な設計とバザールによる品質のトレードオフ**でゲソ……』
「大事なのは機器全体の設計だよ。OS はただの部品にすぎない。本当に OS に問題があるなら NetBSD から Linux に入れ換えてしまえばいいよ。まずは NetBSD の真価を確かめてみて、良い部品であったなら私達の力で BSD 再興というのも面白いんじゃない？」

## 1.2 機能の開花と膨張

『夏はガリガリ君でゲソ!』
「ガリガリ君、いいよね。安くて美味しい。」
『仕事は順調なのカ？』

[*1] https://cruel.org/freeware/cathedral.html

「今のところはね。」

幸運にも私達に割り当てられた仕事は「既存機種同等の起動プロセスを作ること」だった。一番の大仕事は NetBSD の起動プロセスを GRUB のような多段構成に書き換えること。これはこれで骨がおれる作業だが実際にはそれほど大変な仕事ではない。

なぜなら起動プロセスは入口は複数あれど出口は 1 つしか存在しないからだ。入口の例としては外部ストレージ/内部ストレージ/ネットワークからの起動が挙げられるだろう。一方で出口は 1 つしかなく、それは NetBSD カーネルの起動だ。[*2] 出入口が数えるほどしかないのであれば、テストにかかる工数は通常のソフトウェアより大幅に圧縮できる。

『ところでそんなに起動プロセスを改造してしまって、NetBSD オリジナルのソースコードに追従できないんじゃなイカ？』
「そこは私も考えたわ。先輩には、GRUB ベースで作った方が安定して早く仕上ると思います、って伝えたんだけど……」
『だけど、でゲッソ？』
「GPL ライセンスは企業では採用できないって、さ。」
『うーん。GPL ライセンスの沼に飛び込んでしまえば住み心地の良い海が広がっているのにーっでゲソ!』
「まぁ起動プロセスへの要件は完全にこの会社独自のものだし、これはフォークして機能追加し続けるしかないと思うよ。」

またある時は機器の起動時間を短縮する施策を行なっていた。

『どうやって起動時間を短縮するんでゲソ？』
「この機器はアプリケーション同士が相互接続を試みて、両者が接続可能になったらその接続が成立するじゃない。ということはアプリ A とアプリ B の間には相互接続の待ち合わせの方向によって優先度が割り当てられることになるの。」
『するとでゲッソ？』
「もしアプリ B の方がアプリ A よりも優先度が高ければ早めにストレージから読み込んで、なるべく早く起動してあげた方が良いことになる。おそらくこの優先度の関係は、起動毎に変動せず、ツリー状になっているから根のアプリから順番に読み込んで起動してあげるのが理想ね。」
『どうやって実装するのでゲソ？』
「もちろん既存の init プロセスを改造することになるわ。」
『また NetBSD オリジナルとの乖離が発生するんじゃなイカ？』
「そこは……あれ誰か来たみたい。仕事しなくっちゃ。」

機能開発は楽しい。要求が実装となり具現化する。特に楽しいのが機能追加だ。私はモノを改造するのが好きだった。ゼロから何かを作れと言われたら私は硬直してしまう。しかし要求が明確化されていて、既存の実装があれば話は別だ。既存の実装を読み解き理解し、その意思を受け継いで、私の願う正しい方向へと新しい枝をのばす。

こうやって私達は数多くの機能を NetBSD ソースコードをベースに開発していった。しかし先にも触れた通り NetBSD 本体へ還元するには至らなかった。

[*2] 起動に失敗したらエラー画面を表示する必要がありますが今回はその話は省略しています。

# 第2章

# The Heyting algebra ゆ

— Lynn @chordbug

**Yukari** Hey, you know how “programs of a type” and “proofs of a theorem” are really the same thing under some mathematical equivalence?

**Yui** That’s... what I talked about over lunch yesterday.

**Yuzuko** Ack! Stop! I’ll get hungry for curry again!

**Yukari** Yuzuko-chan, do you remember the correspondence?

**Yuzuko** Um, so types are propositions, and showing a type is inhabited by some value is proving it. The empty type Void is falsum.

**Yui** Wow, so you were really paying attention... Anyway, yeah, here’s the full mapping:

| Proof-world | Haskell type-world |
|---|---|
| proof of $a \wedge b$ | value of `(a, b)` (product type) |
| proof of $a \vee b$ | value of `Either a b` (sum type) |
| proof of $a \to b$ | value of `a -> b` (function type) |
| trivial proof of $\top$ | unit value `() :: ()` |
| $\bot$, which has no proofs | empty type `Void` |

表 2.1: The delicious curry isomorphism.

**Yuzuko** What about ¬?

**Yui** $\neg p$ is equivalent to $p \to \bot$, so it becomes **p** → **Void**.

**Yukari** I tried to write some proofs of laws like $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ as Haskell programs, but I got stuck.

**Yui** Let me take a look.

```
type (\/) = Either  -- looks like bunny ears!

dm1 :: ((p \/ q)->Void) -> (p->Void, q->Void)
dm1 f = (f . Left, f . Right)  -- yay!

dm2 :: (p->Void, q->Void) -> ((p \/ q)->Void)
dm2 (pv, qv) = \e -> case e of
                       Left p -> pv p
                       Right q -> qv q  -- wahoo!

dm3 :: (p->Void) \/ (q->Void) -> ((p, q)->Void)
dm3 e = \(p, q) -> case e of
                     Left  pv -> pv p
                     Right qv -> qv q  -- wooo!

dm4 :: ((p, q)->Void) -> (p->Void) \/ (q->Void)
dm4 f = undefined  -- this ones hard??
```

**Yukari** I can't construct a value like **Left f**, or **Right g**...

**Yui** That's right. That one's impossible. In fact, the law is only true in classical logic.

**Yuzuko** What, like, Aristotle?

**Yui** No... well, kinda. Classical logic is just the logic we're used to.

**Yuzuko** Of course. It's called classical logic because it's taught in class.

**Yui** S-sure. But our programs-as-proofs correspondence above is to proofs in *constructive* logic. Basically, that's when you base the logic on things like: "a proof of $p \vee q$ is either a proof of $p$ or a proof of $q$", rather than defining it from truth tables or axioms.

**Yukari** Oh, defining $\vee$ like that is just like defining `Either` myself in Haskell!

**Yuzuko** Is it really that different in practice?

**Yui** Yeah. Because now you can't prove $\forall p\colon\ p \vee \neg p$.

**Yuzuko** Oh... you can't implement `yui` **:: forall p. Either p (p → Void)**.

**Yui** Hey! Don't name it after me—

**Yukari** With `yui`'s help, I could write:

```
dm4 :: ((p, q) -> Void) -> (p -> Void) \/ (q -> Void)
dm4 f = case yui of
          Left p -> Right (\q -> f (p, q))
          Right pv -> Left pv
```

**Yui** I... I guess. Please call it `lem` or something.

**Yukari** Yay! Well, how do I prove that I can't actually prove... it?

**Yuzuko** Can we prove there's a proof you can't prove this proof?

**Yukari** Proo... oof.

# 第3章

# ハイティング代数「ゆ」

— Lynn @chordbug(**翻訳：**@dif_engine)

（この記事は第 2 章を日本語訳したものです。）

**縁** ねえ「ある型のプログラム」と「ある定理の証明」は、どういう意味で数学的に同等なのかわかる？

**唯** それ…私が昨日のお昼ごはんのときに話したやつ。

**ゆずこ** あー！ だめー！ またカレー食べたくなった！

**縁** ゆずこちゃん、対応覚えてる？

**ゆずこ** えーと、型は命題で「ある型になんらかの値が属することを示す」のはその命題を証明すること。そして **Void** が空型。

**唯** おー、ちゃんと聞いてたのか…まあ、あれだ、対応を書くとこうなるな：

| 証明の世界 | Haskell 型の世界 |
|---|---|
| $a \wedge b$ の証明 | 積型 `(a, b)` の値 (product type) |
| $a \vee b$ の証明 | 直和型 `Either a b` の値 |
| $a \to b$ の証明 | 関数型 `a -> b` の値 |
| $\top$ の自明な証明 | Unit 型の値 `() :: ()` |
| $\bot$, これは証明を持たない | 空型 `Void` |

表 3.1: 美味しいカレーの同型対応

**ゆずこ** ¬ はどうなるの？

**唯** $\neg p$ は $p \to \bot$ と同値だから、**p** → **Void** になる。

**縁** Haskell のプログラムとして $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ みたいな法則を証明しようとしたけど、行き詰まっちゃった。

**唯** どれ、みせて。

```
type (\/) = Either  -- うさ耳みたいでかわいい！

dm1 :: ((p \/ q)->Void) -> (p->Void, q->Void)
dm1 f = (f . Left, f . Right)  -- いえい!

dm2 :: (p->Void, q->Void) -> ((p \/ q)->Void)
dm2 (pv, qv) = \e -> case e of
                       Left p -> pv p
                       Right q -> qv q  -- よし!

dm3 :: (p->Void) \/ (q->Void) -> ((p, q)->Void)
dm3 e = \(p, q) -> case e of
                     Left  pv -> pv p
                     Right qv -> qv q  -- わっしょい!

dm4 :: ((p, q)->Void) -> (p->Void) \/ (q->Void)
dm4 f = undefined  -- こいつむずい？？
```

**縁**　**Left f** みたいな値も **Right g** みたいな値も作れない…

**唯**　そう。そいつは証明できない。古典論理だけで真の命題だからね。

**ゆずこ**　古典ってアリストテレスの論理学みたいなやつ?

**唯**　いや…まあそんな感じ。古典論理は要するに私たちが慣れている論理。

**ゆずこ**　そうか。クラスで教わるやつだからクラシカルってわけね。

**唯**　ま、まあな。でもいまの「証明としてのプログラム」対応は**構成的**論理での証明に対するものなんだよ。基本的に、論理を真理表や公理で定義するというより「$p \vee q$ の証明とは $p$ の証明または $q$ の証明である」という原則で論理を考えるわけ。

**縁**　あら、そんなふうに $\vee$ を定義するってのは Haskell で `Either` を自分で定義するのと似てるー！

**ゆずこ**　実際にそんなに違うもん？

**唯**　うん。$\forall p$: $p \vee \neg p$ は証明できない。

**ゆずこ**　おー、たしかに `yui` **:: forall p. Either p (p → Void)** は実装できない。

**唯**　おい！ アタシの名前をつかうな—

**縁**　`yui` ちゃんのおかげでこんなふうに書けました：

```
dm4 :: ((p, q) -> Void) -> (p -> Void) \/ (q -> Void)
dm4 f = case yui of
          Left p -> Right (\q -> f (p, q))
          Right pv -> Left pv
```

**唯**　まあ…そうなるな。てゆーか `lem` とかにしてくれ。

**縁**　やったー！　それで、どうやったら「それが証明できないこと」を実際に証明できるのかな…？

**ゆずこ**　この定理が証明できないことの証明が存在することを証明できるのかな？

# 第4章

# 超実数の実在性について：IST入門

— margo（編集：@dif_engine）

### 編集者による前書き

本記事は「参照透明な海を守る会」の git リポジトリに margo という名義人によってプッシュされていた latex ファイルを元にしたものです．最初のコミットログは”確かにわたしはここにいた”でした．このコミットの約 1 分後のタイムスタンプで”このファイルの扱いはお任せします。Б .”というログが残されていました．サークルメンバーに尋ねてみましたが，奇妙なことに誰も margo 氏や「Б .」という署名をした人物に覚えがありませんでした．それ以来 margo 氏からのプッシュも連絡もありません．

念の為ファイルの内容をチェックすると超準解析に関連する記述が含まれていることが判明しました．私 (dif_engine) が超準解析をある程度理解していたため，この記事のことは私に一任されました．

記事は大きくわけて二つのパートからなっており，最初のパートではその記事自体が書かれるに至った奇怪にして面妖なる経緯が述べられています．次のパートでは超準解析の解説がなされます．とはいえ，お読みになればわかるようにこれらの二つのパートは奇妙な形である程度関連を持っているようです．

超準解析が述べられたパートは，超準解析の成書や論文とはまた違った入門的な読み物としてある程度の意義を持つと思います．

プッシュされていたファイルの体裁を整え，読者の便宜を図って解説や参考文献を付け加えるなどした他はなるべく原型を保つようにしました．

@dif_engine

## 4.1 この記事が書かれた経緯

「ドカッ！」衝撃を受けてわたしは宙を舞った．

「バキッ！」世界が終わる音が響いた．わたしは死んだ．

こうしてわたしのスイートな人生は幕を閉じたわけだが，実のところこの文章の書き手の《わたし》はまさにこの瞬間に生まれたということになるらしい．

春の夜だった．折悪しく小雨が降り始める中，わたしは大学の門から続く交差点を自転車で渡ろうとしていたのだった．側方から猛スピードで走ってくる車には全く気づいていなかった．

わたしの体はぺちゃんこ——まあたぶんそんな感じ——になった．

次の瞬間わたしはベッドの中にいた．わたしは慌てて起き上がり，どこかの骨が折れてやしないかとひとしきり調べた．どこにも異常はなかった．

では結局あれは夢だったのだ——それにしても生々しい夢だったな，などとのんきな事をぼんやり考えていた．やがて意識がはっきりして来ると自分が見知らぬ場所にいることに気づくようになっ

#### 4.2.3

上に述べたように超準解析の導入方法は現在では一通りではないが，所謂《上部構造》方式と呼ばれる典型的な道筋に従うならば次のような道筋をたどることになる．まず実数の集合 $\mathbb{R}$ やそれより大きな基礎集合 $U$ から出発して集合 $\mathbb{U}$ を次のように導入する：

$$
\begin{aligned}
&U_0 := U,\\
&U_{k+1} := U_k \cup \mathscr{P}(U_k) \quad (k \geq 0).\\
&\mathbb{U} := \bigcup_{n\in\mathbb{N}} U_n.
\end{aligned}
$$

その後添字集合 $I$ に関する $\mathbb{U}^I$ のある部分集合を $I$ 上の超フィルターで割ったものを $\mathbb{W}$ と呼ぶ．次に写像 $*\colon \mathbb{U} \to \mathbb{W}$ を構成し，この写像が $\mathbb{U}$ についての文章を $\mathbb{W}$ についての文章に，その真偽を保ったまま移すこと（移行原理）を示す．最後に添字集合 $I$ とその上の超フィルターを適切に構成することにより，$\mathbb{W}$ が好ましい性質を持つようにする．これらの議論は比較的長く退屈であり，読者がその後の一生に渡って使わないであろう技術的な話題も含まれている．そして，超実数が導入されるのはこれらの議論が済んだあとなのである．

もしこのように面倒な議論が不可欠だとしたら，超実数に対して実在感を持つことは難しいだろう，と言わざるを得ない．だが本当にそれは超準解析を**理解する上で**不可欠なのだろうか？

#### 4.2.4　数学の《形式的理解》と《使用》について

そもそも，我々が何らかの数学的概念を自在に使いこなせるようになるということと，その概念の構成を理解することはどのように関係しているのだろうか．あるいは——全く関係していないとは言えないまでも——関係が浅い可能性を否定できないのではないだろうか？

この問題について考えるために，一時的に超実数の話題から離れて実数について考えてみよう．それなりの高等教育を受けた人間にとって，実数の概念は己の手指のように身近なものである．一辺が 1.0m の正方形の対角線の長さは $\sqrt{2}$m であり，この $\sqrt{2}$ という数を十進法で表示しようとすると 1.41421356… という際限なく続く数字の列が得られること，この数が $x^2-2=0$ という代数方程式の解の一つであること，実数のなかにはいかなる代数方程式の解にもならないような——超越数と呼ばれる——ものがあること，といった話に我々はすっかり馴染んでいる．このような意味で実数は確かに身近な概念である．このように実数を身近に感じるためには《実数とは**何であるか**》を基礎から——典型的にはデデキントの切断やカントールの基本列を用いて——講義されねばならないわけではない，ということに読者の皆さんは自らの経験に照らして同意していただけることと思う．

実数についてここまで述べてきたのは，つまるところ数学的な概念を巡る《形式的な理解》と《使用》の違いを例示したかったからである．数学的概念の形式的理解の価値を軽く見るわけには行かないが，我々がある数学的概念をしっかりと把握するためにはその使用に習熟することが不可欠である．そして，そのような習熟によってこそ，我々はその数学的概念に対して確かな信頼感を持つことができるのである．

#### 4.2.5

だからといって，我々はオイラーのように《無限大》や《無限小》の量を素朴に考えて闊達に扱うわけにはいかない．というのもこれらの概念の《素朴な》バージョンは現代では常識的と思われるレベルの批判に耐えないからである．

念のため，このことを《無限大の自然数》を例にとって説明しよう．もし我々が《無限大であるような自然数 $n$》を認めるならば，よく知られた自然数の性質により《最小の無限大自然数 $m$》が存在しなければならない．すると $m$ の最小性により $m-1$ は無限大ではない．しかしながら，このよ

(Rule1)　$0 \in \mathbb{N} \cap \mathbf{S}$.
(Rule2)　$\forall n \in \mathbb{N} \cap \mathbf{S}\ \ (n+1 \in \mathbb{N} \cap \mathbf{S})$.
(Rule3)　$\exists n \in \mathbb{N} - \mathbf{S}$.
(Rule4)　自由変数を一つだけ持つ論理式（外的でも良い）$A$ に対して
$(A(0) \wedge \forall n \in \mathbb{N} \cap \mathbf{S}\ \ (A(n) \implies A(n+1))) \implies \forall m \in \mathbb{N} \cap \mathbf{S}\ \ A(m)$.

(Rule1) は，0 が標準的な自然数であることを述べている．
(Rule2) は，$n$ が標準的な自然数であるとき，$n+1$ も標準的な自然数であることを述べている．
(Rule3) は，超準的な自然数が存在することを述べている．
(Rule4) は，$\varphi(0)$ であり，任意の標準的な自然数 $n$ に対して《$\varphi(n)$ ならば $\varphi(n+1)$》であるとき，《任意の標準的自然数 $m$ に対して $\varphi(m)$》であることを述べている．

#### 4.2.10　$\mathbb{N} \cap \mathbf{S}$ が真クラスであること

(Rule1)(Rule2) を組み合わせれば，

$$0, 1, 2, 3, 4, 5, 6, 7, ....$$

のような自然数がすべて標準的であることが観察される．
そこで読者は，数学的帰納法により次が証明できると予想するかもしれない：

(間違った予想)　$\forall n \in \mathbb{N}\ \ n \in \mathbb{N} \cap \mathbf{S}$.

上の予想について考えるために，形式的自然数に対する数学的帰納法がどのようなものであったかを思い起こしておこう：

**形式的自然数の集合 $\mathbb{N}$ に対する数学的帰納法**
任意の集合 $K$ に対し次が成立する：

$$(0 \in K \wedge \forall n \in \mathbb{N}\ \ (n \in K \implies n+1 \in K)) \implies \mathbb{N} \subseteq K.$$

よって，もし $\mathbb{N} \cap \mathbf{S}$ が集合ならば，上記の数学的帰納法によって確かに $\mathbb{N} \cap \mathbf{S} = \mathbb{N}$ となる．しかしその場合 (Rule3) が成立しない．したがって，$\mathbb{N} \cap \mathbf{S}$ は集合ではあり得ず，真クラスでなければならないことがわかる．

読者は $\mathbb{N} \cap \mathbf{S} \subseteq \mathbb{N}$ なのに $\mathbb{N} \cap \mathbf{S}$ が真クラスであることを奇妙に感じるかもしれない．ZFC 集合論においては，真クラスは《巨大すぎる》ために集合になれないクラスであると，しばしば漠然と理解されている．我々が追加した公理および公理図式の元では《ZFC の言語での記述が及ばない》ため集合になれないクラスが存在すると考えると良いかもしれない．以上の議論から，クラス $\mathbf{S}$ が真クラスでなければならないことも従う．というのも，もし $\mathbf{S}$ が集合ならば $\mathbb{N} \cap \mathbf{S}$ も当然集合となるはずだからである．

一方で，$\mathbb{N} \cap \mathbf{S}$ に対して，数学的帰納法の代替物となるのが (Rule4) である．

#### 4.2.11　超準自然数

$\mathbb{N} - \mathbf{S}$ の元を**超準自然数 (nonstandard natural number)** と呼ぶことにしよう．
超準自然数に関して次が成立する：

**命題 4.2.1**　任意の $\nu \in \mathbb{N}$ に対して次の (1)(2) は同値である：