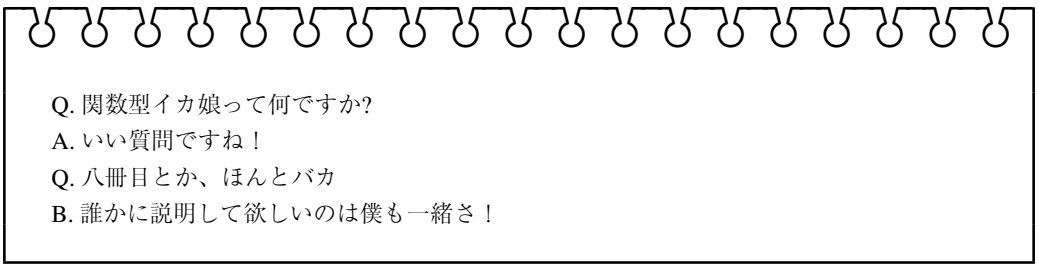
第0章

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. 八冊目とか、ほんとバカ
B. 誰かに説明して欲しいのは僕も一緒さ！

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。

この本は八冊目の関数型イカ娘の本でゲソ。シリーズも後半に突入したので、少し雰囲気を変えていくでゲソ! アニメ 3 期の放映が近いことを祈りつつ、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。ただ、一般的な入門書ではないでゲソ。

第1章

エルエフ島漂流記

— @master_q

1.1 不時着

今日は何時でここは何処だろう。とにかくこの小さな島は僕が住んでいた島と陸続きではないようだ。僕等はまだ見ぬ新しい技術を手に入れるために祖国をはなれて船出したのだ。しかしあのひどい嵐に飲まれてこの島に不時着してしまった。計画にない出来事に僕の頭は混乱していた。
「で、この後どうするんでゲソ？」
船を調べてみたところ舵が完全に壊れてしまっている。この状況では今日や明日に船を出すのは不可能だ。どうやらこの島である程度の期間、生活する必要があるようだった。

僕等の船のソフトウェア実装にはできうるかぎり Coq [*1] で証明が付けてあった。船はクリティカルなシステムなので、何らかの方法で安全性を担保する必要があるのだ。ところが、この島では Coq ではなく ATS/LF という証明器しか使えないらしい。そんなわけで僕は ATS/LF について少し調べてみることにした。

1.2 ATS/LF とは

ATS/LF の概要を知るために、まずはそのマニュアルである「ATS プログラミング入門 [*2][*3]」を読んでみた。

それによると、ATS/LF は ATS 言語 [*4] の持つ定理証明サブシステムの名前で、全域関数を使って証明を構築する。証明に使うこの全域関数は証明関数という実行される実体を持たない特殊な関数で、通常の関数と異なりコンパイル時に型検査されるためだけに存在する。カリー＝ハワード同型対応によると計算機プログラムと証明との間には直接的な対応関係があり、「プログラム＝証明」「型＝命題」のように対応づけられるが、まさしくこの対応によると ATS 言語においては関数と型は表 1.1 のような対応を持っていることになる。

動的	型 :=	`fun` キーワードで宣言される関数シグニチャ
	プログラム :=	`implement` キーワードで定義される関数本体
静的	命題 :=	`prfun` キーワードで宣言される証明関数シグニチャ
	証明 :=	`primplement` キーワードで定義される証明関数本体

表 1.1: ATS におけるカリー＝ハワード同型対応

[*1] `https://coq.inria.fr/`
[*2] 英語原著: `http://ats-lang.sourceforge.net/DOCUMENT/INT2PROGINATS/HTML/`
[*3] 日本語訳: `http://jats-ug.metasepi.org/doc/ATS2/INT2PROGINATS/`
[*4] `http://www.ats-lang.org/`

このままでは証明とプログラムは完全に分離されていて関与することができないが、ATS/LF ではプログラムと証明を混ぜて書き下すことができる。関数インターフェイスの意味として関数シグニチャに適切な命題を付記し、関数本体で型と命題が同じ意味になるようにプログラムと証明を混ぜてプログラミングをすることで、証明とプログラムの性質が一致することを期待できる。

少しわかりにくいので整理しよう。結局のところ ATS における関数は次の 3 種類に分類できる (図 1.1)。

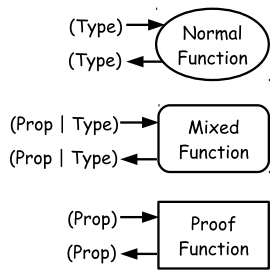


図 1.1: 関数の種類

通常の関数　引数と返り値は型で表わされる。これは他の関数型言語の関数と同様のもの。

混合の関数　引数と返り値は型と命題のタプルで表わされる。これは他の関数型言語の関数に命題が付記されたもの。

証明関数　引数と返り値は命題で表わされる。

これら 3 種類の関数はお互いを呼び出すことができるが、証明関数だけは通常の関数や混合の関数を呼び出すことはできない。これらの関数はお互いを呼び出せるので、実際のアプリケーションはこれらの関数を組み合わせて構築することができる。例えばプログラムのエントリポイントは多くの場合は通常の関数だ。なぜなら POSIX においてプログラムの実行開始と終了には何ら証明的な制約がないからだ。しかしアプリケーションから使うライブラリの呼び出しにおいては、しばしば証明的な制約をつけたくなる場合がある。このような場合、ライブラリ関数のインターフェイスでは混合の関数を使うことがある。この混合の関数を通常の関数から呼び出すと返り値に命題が返されることがある。このような命題は通常の関数本体で束縛しておくことで、別の混合の関数の引数に渡すこともできる。

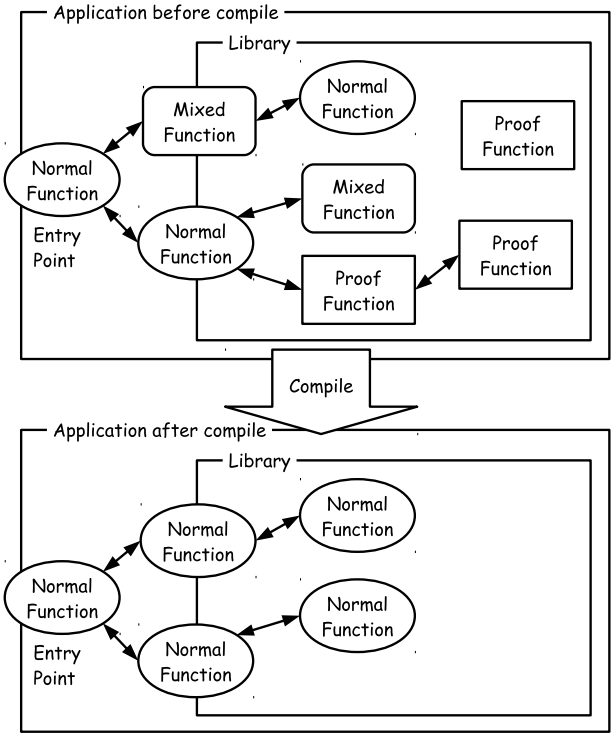


図 1.2: 3 種の関数を組み合わせたアプリケーション

ライブラリ内部においては静的な性質を証明するために、しばしば証明関数を呼び出すことがある。この証明関数は実行時には実体を持たないが、プログラム中の命題に対する関数的な操作を行なうことができる。また、どこからも呼び出されない証明関数も時に有用で、これは証明関数が表わす命題に対して証明を与えたい場合に使われる。

このような 3 種の関数を組み合わせたアプリケーションはコンパイルされると内包する証明コードが型検査された上で削除される (図 1.2)。ATS における証明コードは実行バイナリ中に実体を持

第2章

二重否定と継続

— @nushio

2.1 悪魔の契約

『契約の獣キュゥべえ (以下甲) と少女 (以下乙) は以下のように契約する。

(1) 甲は乙に 1 億円支払う、または、(2) 乙が甲に 1 億円支払えば、甲は乙の願いを何でも叶えてあげる、ただし、(1)、(2) の選択権は甲が有する。甲乙の間に、魂の引き渡しや加工等といった、上記以外の権利義務お節介は発生しない』

などという契約をさせる詐欺が流行っているそうなので、皆さん気をつけるように。さて、先に進む前に、まずは考えてみたい。あなたが少女の立場だとして、この契約に価格をつけるとしたらいくらの価値があるだろうか？

常識的な解釈では、次のようになるのではないか。(1)、(2) の選択権はキュゥべえにあるわけだが、もしキュゥべえが (1) を選択してきたなら、あなたはタダで 1 億円を手に入れる。(2) を選択してきたなら、なんとかして 1 億円を調達しないといけないが、その見返りは途方もなく大きい。少なくとも、借金をして 1 億円を調達し、2 億円と利子を願うことができる。したがってこちらのオプションの価値も 1 億円以上ということになる。借金が無理ならもっとやましい手段で資金を調達したうえで、その証拠も記憶も、完全に消してしまうこともできるし、望むなら全能の神にだってなれるわけだ。

ところが、実はこの契約には一銭の価値もないのだ。キュゥべえは何の自己負担もなく、この契約を履行してしまえる。後には『願いを何でも叶えてあげる』という甘言に釣られた、少女の後悔だけが残るだろう。そして第二次性徴前の少女が 1 億円もの大金を調達するとなると、その手段は大きな後悔を伴うものに限られるに違いない。

エロ同人みたいに！

エロ同人みたいに！

キュゥべえはいったいどのような手を使ってこの契約を履行するのか？ このような甘言を振りまいて、少女を絶望に陥れるキュゥべえから身を守るにはどうすればいいのか？

それを、プログラミング言語と型の理論から解き明かしていくのが、今回のテーマだ。

2.2 悪魔のプログラム

この『契約』を利用する戦略は、プログラムで書けば以下のようになるだろう。

```
useTheContract :: Either Money (Money -> IO' Void) -> IO' ()
useTheContract c = case c of
  Left money -> do
    story $ "さやかちゃん：いきなりお金を手に入れた。ラッキー！ " ++ show money
  Right qbey -> do
```

```
  story "さやかちゃん：お金を工面してキュゥべえに渡すよ。"
  (g :: Void) <- qbey Money
  story   "さやかちゃん：神になったよ！ 自分で自分を褒めてあげたい感じ？ "
  story $ "さやかちゃん：とりあえずお金を倍に増やして借金返して: "
       ++ show (absurd g :: Money)
       ++ show (absurd g :: Money)
  story $ "さやかちゃん：それから満漢全席食べるよ！ : "
       ++ show (absurd g :: MankanZenseki)
```

このプログラムは、まえにのべた契約に価格をつける戦略を Haskell のプログラム化したものだ。ここで、`Either a b` は a 型または b 型の値のどちらかが入っている型だ。また、`Void` とは、hackage にあるライブラリ `void` に属するモジュール `Data.Void` の提供する型である。`Void` を特徴づけるのは、次のような関数 `absurd` の存在だ。

```
absurd :: Void -> a
```

信じがたいことに、`Void` 型の値がひとつでもあれば、それを `absurd` に適用することで、どんな型の値でも作り出すことができるのだ。[*1]

以上を踏まえれば、上記のプログラム `useTheContract` の引数の型 `Either Money (Money -> IO' Void)` が、くだんのキュゥべえの契約に対応しているという気分がわかってもらえるだろうか。`IO'` というのは IO モナドの機能を拡張して継続が使えるようにしたものだ。継続とは何か... それが今回の話のネタなので、解説をお待ちいただきたい。

では、このプログラムを実行してみよう。じつは、この契約を提供するキュゥべえ側は、以下のプログラムにより、この契約を履行することができる。

```
type Contract = Either Money (Money -> IO' Void)

offerContract :: IO' Contract
offerContract = callCC f
  where
    f :: (Contract -> IO' Void) -> IO' Contract
    f k = do
      story "キュゥべえ：ボクは (2) を選ぶよ。"
      return (Right (\x ->
        story "キュゥべえ：ボクは (1) を選ぶよ。"
          >> k (Left x)))
```

キュゥべえとさやかちゃんを対話させるには、以下のコードが必要だ。

```
{-# LANGUAGE ScopedTypeVariables #-}
import Control.Monad
import Control.Monad.IO.Class
import Control.Monad.Cont
import Data.Void
```

[*1] このような `Void` だが、何の問題もなく Haskell のプログラムとして実装することができる。気になる方はライブラリ `void` のソース、もしくはこの章の最後にある独立版を参照いただきたい。

第3章

真の名を〜データ設計とAlloy〜

— @osiire

ことばは沈黙に 光は闇に 動は静のなかにこそ あるものなれ
飛翔せるタカの 虚空にこそ 輝ける如くに
– エアの創造より

3.1 プロローグ

3.1.1 困難な課題

ローク島の魔法学院では豊穣の祭りに向けた準備が慌しく進められていた。今年は魔法使いとしても名高いオー島の領主とその美しい妃が招待される予定とあって、院生達はいつになく張り切っている。夕食会のディナーのために特別なワインが揃えられ、広間のセッティングには様式の長までが自ら進んでたいまつの配置から天井の飾りにまで事細かに指示を出していた。一方、学院から離れローク島最北端の岬に建つ隠者の塔では、名付けの長が書いては消す大量の真の名を今日も 7 人の院生が黙々と書き写し覚えている。その隠者の塔の一室で、ハイタカは大量の書き込みがある縒れたノートを前に一人焦っていた。名付けの長に出された課題に対する作業が遅々として進まなくなっていたからである。

「塔の中にある物の真の名を全て列挙し、関連性を示せ」

そう長から言い渡されたのは、塔からうっすらと見えるまぼろしの森の木々の濃い緑が映えていた頃であった。

「隠者の塔はそう広くない。真の名などすぐに列挙できるだろう。」

未熟な 4 年生はタカをくくっていた。しかし、実際に課題に取りかかると、その困難さはすぐに分かった。まず、物の真の名を探り当てる事が意外と大変だった。既に名を知っている物は簡単だが、真の名を知らない物については教科書や古代文献を探さねばならない。その上で、調べた真の名を呼んでみてまやかしの術から開放されるかどうかなど、実験して確かめる必要がある。あばかれていない真の名の場合、それを探り当てるのに魔法使いが一生を費やすこともあるくらいである。塔の中にそこまで難しい物ははないとはいえ、探り当てに躓くとそれだけで一週間浪費することもあった。

さらに、名を列挙していく内に、名同士には一対多/多対多などといった関係がある事が分かった。例えば、テーブル (真の名をトーニと言う) には必ず一枚の天板と一本以上の足があるといった風に。これが長が指示していた関連性である。そうなると、いくら狭い塔の中とはいえ示さなければならない関連性の組み合わせは 100 や 200 ではなくなる。関連性の発見と記述はどんどん膨大な物になっていき、提出予定のノートは関連性の線が幾重にも交差して黒くなる一方だった。

この気の遠くなるような課題、提出期限は冬至までである。もう残された時間は少ない。こんな

課題ができないようではヒスイにも勝てない！ カラスノエンドウは今頃どうしているであろうか。どうにかこの状況を打開できないか。ハイタカの頭の中では、何度も同じ言葉が浮かんでは消えていた…。

3.1.2 師の教え

海から吹く風がずいぶん寒くなってきたある日、食堂で遅い昼食を食べ終えたハイタカは塔への戻り道の途中の道端に、タラクサカムの一種が生えているのを偶然見付けた。

「珍しい。オジオンに教えてもらったことがある、薬草としても使えるカントウタラクサカムだ。こんなところでも自生しているのか。」

その黄色い小さな花と幾重にも重なったギザギザの葉を膝をついて眺めながら、彼はゴンド山で師と過ごした日々を思い出していた。オジオンは山中の草木の効能や名前 (真の名も) を丁寧に教えてくれたものだった。

師は偉大な魔法使いだ。しかし、さすがのオジオンでも記憶力に限界があるはず。どうやってあんな大量の名を覚えていたんだろう。老いた魔法使いの後ろ姿を想像しながら、そんな疑問がふと浮かんだ瞬間、思い出した。

「そうだ、Alloy だ！ オジオンは沢山の草木の名を Alloy Analzyer で整理していた。あれを使えば塔の名もきっとうまく整理できるに違いない！」

思わず立ち上がって叫んでいた。ハイタカは早速、今しがた出てきたばかりの食堂の横を抜け、隠者の棟に比べると一際明るい電算棟に駆け込んだ。

3.2 Alloy の基礎

3.2.1 Alloy とは

電算棟の自習室には 10 台近くの平机が所狭しと並べられ、平机一つにディスプレイと端末一つが配置されていた。それぞれの机の下ではゴミ箱のような大きな端末が唸りをあげ、部屋の天井に這っている黄色いケーブルと何かを送受信していた。中には”Macintosh”と書かれたディスプレイ一体型の小型端末もあったが、彼は使い方がよく分からなかったので近寄らないことにしていた。ハイタカは、大きく”X”と描かれた画面の前に座り、真の名を入力してログインした。

「Ged」

ネットの情報によると、Alloy Analyzer [3](`http://alloy.mit.edu/alloy/`) とは、Daniel Jackson らによって発明された有界モデル発見器というものの一種らしかった。

- Alloy Analyzer は Z に似た記法の言語を持ち、物 (エンティティ) 同士の関係 (Relation) をベースに仕様 (モデル) を表現できる。
- 記述されたモデルから、モデルを満たす例 (インスタンス) を SAT ソルバーを用いてスモールスコープ内で全探索し、図まで示してくれる。(スモールスコープとは、記述された複数のエンティティの数をそれぞれ 0 から一定の数 N まで変化させた時に得られるインスタンスの集合の事。一定の数 N が実質的に 10 程度と大きくないので「スモール」スコープと呼ばれる。)
- Java 製のオープンソースソフトウェア (`http://alloy.mit.edu/alloy/download.html`) であり、Java の実行環境があれば誰でもすぐに利用できる。

これらの性質を持つ Alloy Analyzer は [3] でも述べられている通り、所謂「データ設計」に応用できるようだ。データ設計では物 (エンティティ) やその属性に名前を与えてその構造を決める。関係の記述が得意である Alloy Analyzer は、これらの構造を簡潔かつ正確に記述できるという訳だ。しかも、Alloy Analyzer が示してくれるインスタンスや無矛盾性チェックは、設計をセルフチェックするために利用できる。

第4章

Lagrangeの未定乗数法

— @dif_engine

物語の概要と目的

現実世界における「滑らかな問題」の最適解の候補を見出すための手法として応用上重要なLagrangeの未定乗数法について学びます。この記事では線形代数や微積分の復習をしながら、ゆったりとしたペースでこの手法について学びます。

4.1 プロローグ

魔理沙が遊びに来ている。一時期はしきりにHaskellについての質問をしてきたが、最近は学習理論に興味があるらしい。

「——でさ、Lagrangeの未定乗数法ってのがでてきてさ」

「……ええ、学習理論の本を読んでいると、分布の最尤パラメータを求める場合にLagrangeの未定乗数法を使ってる事が多いわね」

「そうそう、でもあれって一体なんでうまくいくのかよくわからないんだよな」

「変数が少ない場合の話はわかるかしら？」

「等高線と拘束条件の曲線が接してるから……みたいな絵で説明してるのは見たことあるぜ。二変数のときはあれで一応わかったことにしてるんだけどさ」

「次元がもっと高い場合の話がわからない……？」

「そうそう！　高次元でも同じだよ、とか言われても高次元はうまく思い浮かべられないぜ」

「高次元への直観が働かないから、高次元への直観を前提とした説明も理解できず、ついにはLagrangeの未定乗数法自体が疑わしくなってしまった……という感じかしら」

「それな。パチュリーはやっぱすごいな」

「おだてたって何もでないわよ。さて、ではLagrangeの未定乗数法の確認をしておきましょう。でもその前に最低限の記号の整理をしておきましょう——。」

本記事での約束 (1)

- 自然数 M, N は $1 \leq M < N$ という関係を満たすものとします。N 次元ユークリッド空間 $\mathbb{R}^N$ をしばしば直和 $\mathbb{R}^M \oplus \mathbb{R}^{N-M}$ と同一視します。この同一視に従って、$x \in \mathbb{R}^M$ と $y \in \mathbb{R}^{N-M}$ の対 $\langle x, y \rangle$ を $\mathbb{R}^N$ の元とみなします。
- $\mathbb{R}^N$ の部分集合 Ω 上で定義された関数 F を、直積 $\mathbb{R}^M \times \mathbb{R}^{N-M}$ 上の関数に置き換えたものを $\overline{F}$ で表すことにします：

$$\overline{F}(x, y) := \begin{cases} F(\langle x, y \rangle) & \langle x, y \rangle \in \Omega, \\ \varnothing & \text{otherwise.} \end{cases}$$

- 自然数 $K(\geq 1)$ に対して、
$$\mathscr{O}_K := \{\mathbb{R}^K \text{ の開集合}\}.$$
- 点 $x \in \mathbb{R}^K$ に対して、
$$\mathscr{O}_K(x) := \{P \in \mathscr{O}_K \,|\, x \in P\}.$$
つまり、$\mathscr{O}_K(x)$ は点 x の開近傍全体からなる集合です。
- i 行目で j 列目の要素が $f(i,j)$ として与えられた $M \times N$ 行列を
$$\left[\begin{array}{c|c} f(i,j) & \begin{array}{c} i:1 \downarrow M \\ j:1 \rightarrow N \end{array} \end{array}\right]$$
とも書くことにします。
- 一列しかない行列は
$$[\ f(i)\ |\ i:1 \downarrow M\ ]$$
のように書くことにします。
- 一行しかない行列は
$$[\ f(i)\ |\ i:1 \rightarrow M\ ]$$
のように書くことにします。

「この行列の表記はあまり見たことないな」
「成分の添字のどっちが行でどっちが列だか混乱しそうな場合にはこの記法は便利よ」
「なるほどな……！」

4.2 Lagrange の未定乗数法

「——では Lagrange の未定乗数法のステートメントを書き出してみるわ」

Lagrange の未定乗数法

前提：

- $\Omega :\in \mathscr{O}_N$
- $f, g_1, \ldots, g_M :\ \in C^1(\Omega;\mathbb{R})$
- $V := \{x \in \Omega \mid {}^\forall i \in \{1,\ldots,M\}\ \ (g_i(x) = 0)\ \}$
- $z_0 :\in V$

主張：**(Lag1)** かつ **(Lag2)** のとき **(Lag3)** かつ **(Lag4)**.
ただし、

(Lag1) ${}^\forall z \in V \quad \operatorname{rank}\left(\left[\begin{array}{c|c} \dfrac{\partial g_i}{\partial z_j}(z) & \begin{array}{c} i:1 \downarrow M \\ j:1 \rightarrow N \end{array} \end{array}\right]\right) = M.$

(Lag2) z_0 が $f\big|_V$ の極値点である.

(Lag3) ${}^\exists \lambda_1, \ldots, \lambda_M \in \mathbb{R} \quad f'(z_0) = \displaystyle\sum_{k=1}^{M} \lambda_k\ g_k'(z_0),$

ただし
$$f'(z_0) = \left[\begin{array}{c|c} \dfrac{\partial f}{\partial z_i} & i:1 \rightarrow N \end{array}\right], \text{ etc.}$$