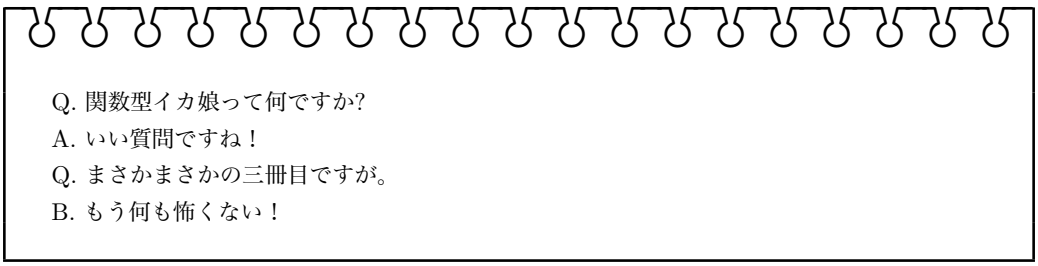
目次

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. まさかまさかの三冊目ですが。
B. もう何も怖くない！

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)」に続く、三冊目の関数型イカ娘の本でゲソ。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。

第1章

蓮物語

— @dif_engine

物語の概要と目的

集合と写像について一応のことを知ってはいるけれど 関数型言語に触れたことがない 主人公が Haskell を学びます。同時に主人公は圏論の初歩を学び、圏論を応用すると Haskell の幾つかの重要な型クラスの挙動が理解しやすくなることを知ります。物語を通し、主人公は最小限の圏論の知識を手がかりに Haskell の型を理解しようと試行錯誤することになります。

このような話題をきちんと論じるならば、集合と写像のなす圏を圏論の立場で丁寧に抽象化してから論じることになるでしょう。とはいうものの、圏論をゼロから学び cartesian closed category のあたりまで進めるのはなかなか大変です。そこまで徹底した抽象化を前提としなくても Haskell と圏論の関わりを説明することはある程度可能だと筆者は考えました。物語を通して、圏論についての初歩的な知識を紹介し、Haskell の幾つかの重要な型クラスが従うルールを圏論の立場から説明します。

1.1 プロローグ

怪異――。

そう呼ぶのが如何にもふさわしいように思えた。髪の毛の代わりに頭から青い触手が生えている少女を怪異と呼ばないのなら怪異とは一体何のための言葉だというのだろう。

晴れた夏の午後だった。他の観光客たちが思い思いにビーチライフを満喫する中、海の家で少し遅目の昼食を食べた僕は場違いにもテーブルに本とノートを広げて思案していた。

「――それは Haskell じゃなイカ？」

食べ終えたカレーライスの皿をテーブルから下げるついでに僕のノートを覗きこんだ少女はそんな風に声をかけてきた。少女はイカ娘と名乗った。

「いか――むすめ？」

「そうでゲソ」

風変わりな姿に似つかわしく、やはり名前も風変わりだと思った。他人の名前についてあれこれ言うのは――しかも初対面の相手に対してならなおさらだ――いささか不作法ではあったが僕は素直にそんな感想を述べた。もしかしたら思いがけず怪異に遭遇したことで僕はいくらか動揺していたのかもしれない。

「阿良々木暦（あららぎこよみ）という名前も十分風変わりじゃなイカ？」

これは僕の名前に対する彼女の感想。

1.3.1 圏論（2）

Haskell：アプリカティブファンクター

昼飯時を過ぎているせいか、この時間になると客もまばらだった。最初に話しかけて少し言葉を交わしたあとも、イカ娘は店の手伝いをしながら僕のテーブルの側で立ち止まっては興味津々といった様子で僕のノートを覗きこんだりしていた。少し——いや、かなり気が散るのだが。それにしても、イカ娘は Haskell がそんなにも気になるのだろうか——。

ビーチではしゃぐ観光客の楽しげな声が耳に入るせいなのか——それとも店内を歩き回るイカ娘の華麗な触手さばきに時折目を奪われるせいなのか——あまり集中できないながら、僕は青い象の本の 11.3 を読んでいた。アプリカティブファンクター（applicative functor）の箇所だ。型クラス `Applicative` の主要な部分はこうだ——。

```
class Functor f => Applicative f where
   pure :: a -> f a
   (<*>) :: f (a -> b) -> f a -> f b
```

『`Functor f =>`』という箇所は『`f` が `Functor` 型クラスのインスタンスであるときだけ `f` は `Applicative` 型クラスのインスタンスになれる』という制約条件を表している。——というわけで、`Applicative` のインスタンスは `Functor` のインスタンスでもあるということになる。
そこまではわかる。
だが——なぜこんなものが必要なのだろうか？　それがよく分からなかった——。

分からないという状態にも色々なレベルがある——分からないポイントが自分で解っているレベルの「分からない」から、分からないのポイントが一体どれなのかがすでに判らないというレベルの「分からない」まで——。
自分が今直面してる「分からない」は多分その真ん中あたりだ。青い象の本にはアプリカティブファンクターはファンクターの強化版だと説明してある——まずここがわからない。——なぜ強化する必要があるのだろう？　もし `Functor` 型クラスのインスタンスが型を対象とし、関数を射とする圏 $\mathbf{C}$ から自分自身への関手だというのなら、可換図式はそのまま可換図式に写るのだから、改めて強化する必要はどこにもないはずだ。

考えていてもよくわからず、集中力が——ますます——低下しているのが自分でもよくわかった。
——気分転換しよう。僕以外の客がいなくなってしまったタイミングで、あいかわらず僕のノートが気になる様子のイカ娘に話しかけてみた——。
「なぁ、Haskell に興味があるのか？」
まるで猫缶を見せられた人馴れした野良猫のように目を輝かせてイカ娘がいそいそとやってきた。
「Haskell になってからよく知らないから興味あるでゲソ！」
『Haskell になる』ってどういう意味だろう——。
まあいいや。
少し話をしてみると、イカ娘が卓越した数学の才に恵まれている事に否応なく気付かされた。彼女はどうやらあまりきちんとした教育は受けていない様子だったが、にも関わらず僕のノートや羽川から借りた本をパラパラと数分間めくっただけで僕の知識と理解に追いつき——それどころか僕が教えを請わねばならない立場になっていた。自分には数学の才能があるなどという自負心を——少しばかり——

第2章

RTS海溝二万マイル

— @master_q

みんな毎日 GHC で元気にはすはすしてるでゲソ？ Hackage には色々なパッケージが登録されていて、それらの使い方を調べると新しい知見が得られ、読んでるだけで楽しいでゲソ。

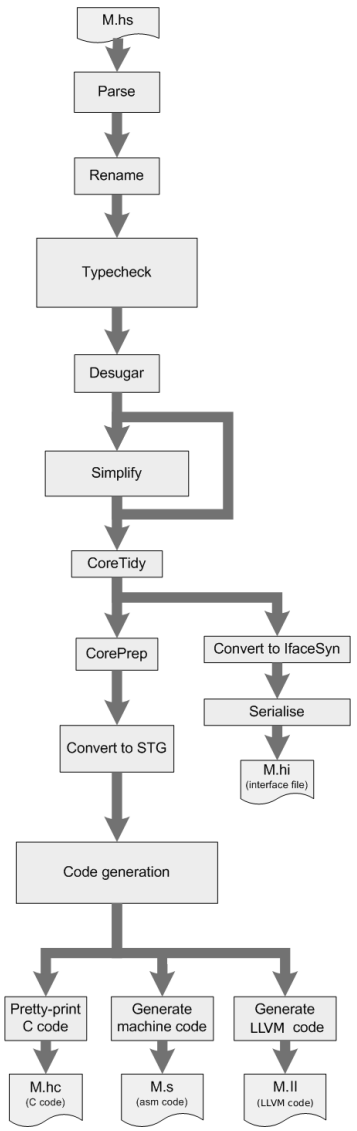
ところで、こんな楽しい世界 (GHC) がどんなしくみで動いているのか知りたくなイカ？

2.1 RTS は Haskell コードを実行する VM でゲソ!

一言に GHC と言っても中は広いんでゲソ。今日はその中でも RTS(Runtime System) というモジュールの中を探検してみようと思うでゲソ。

ところで RTS ってなんでゲソ？ちょっと Haskell から離れて他の言語の実行モデルを思い出してみるでゲソ。C 言語の場合、ソースコードをコンパイラにかけるとアセンブリ言語になり、アセンブラを通すことで機械語になるでゲソ。この機械語同士をリンクすれば実行バイナリができるでゲソ。Java 言語の場合、ソースコードをコンパイラにかけると機械語ではなくバイトコードになるでゲソ。このバイトコードは Java VM という仮想マシンの上で実行できるでゲソ。その Java VM 自体は C 言語や C++ 言語など Java 以外の低レベルな言語で設計されることが多いでゲソ。Perl 言語の場合、ソースコードはコンパイラにかけず、Perl インタープリタ上で実行するでゲソ。そのインタープリタはやはり C 言語のような低レベル言語で設計されているでゲソ。

さて、Haskell に戻ってみるでゲソ。GHC は Haskell のソースコードを右の図のようなコンパイルパイプライン[*1] を通して機械語に変換するでゲソ。他の言語と比較すると C 言語のケースと似ているでゲソね。この機械語化された Haskell の関数をつなげれば、Haskell で作ったプログラムの意図通りに動作をする実行バイナリができるはずでゲソ。じゃあ、Haskell で書かれたソースコードはこのパイプラインを通して



[*1] http://hackage.haskell.org/trac/ghc/wiki/Commentary/Compiler/HscMain

第3章

評価

— @xhl_kogitsune

「お前たち! 今日も元気に簡約するでゲソ!!! タイトルを見て千反田とかいう奴が出てくることを期待していたのなら残念だったでゲソね! ここは私が侵略済みでゲソ!!

今日は式の**評価**の話をするでゲソ! 評価、あるいは**簡約**、というのはここでは計算の 1 ステップみたいなものでゲソ。たとえば式 `3 * (1 + 2)` を 1 回評価 (簡約) すると式 `3 * 3` になって、更に 1 回評価 (簡約) すると値 `9` が得られる、みたいな感じでゲソ。

一言に評価と言っても、やり方は色々あるでゲソ。式のどこから評価をしていくか、という**評価順序**も色々あるし、1 回の評価をどう実装するかも色々あるでゲソ。今日は、私オススメの評価順序である Normal Order 評価の話をしつつ、答えが出るまでにかかる評価回数を減らすにはどのように 1 回の評価を実装すればいいかという話をするでゲソ。

まず式 1 の評価を考えるでゲソ。ここでは Haskell 風に書いたでゲソが、`(\x -> x * x)` という「x を受け取って $x * x$ を返す関数」を $(1 + 2)$ に適用しているでゲソ。

式 1: Haskell

```
(\x -> x * x) (1 + 2)
```

OCaml だと

式 1: OCaml

```
(fun x -> x * x) (1 + 2)
```

C++ だと

式 1: C++

```
int f(int x){ return x * x; }
int main(){
...
int r = f(1 + 2);
...
}
```

こんな感じでゲソ (r の値を計算するでゲソ)。今日は一番最初の Haskell 風表記を使って説明するでゲソ。評価順序とかは実際の Haskell 実装とは違うことがあるので注意でゲソ! 式 1 に出てきた、関数、関数適用 、四則演算の書き方くらい把握しておけばだいたいサンプルコードは読めるんじゃなイカ?

第4章

オール・アバウト・ケーゾク・イン・スキーム

— @yryozo

お主たち！今日も元気に括弧をつけてるでゲソか?

括弧の乱れは心の乱れ、心の乱れは家庭の乱れ、(略)、そして国の乱れは宇宙の乱れ、とどこかの偉い人も言ってたような気がするでゲソ。これからも精進してカッコいいプログラマーを目指すでゲソよ。

さて、前回は Scheme にとって大事な「末尾呼び出しの最適化 (Tail Call Optimization, TCO)」の話をしたでゲソ [*1]。でも Scheme にとって大事なものはもう一つ [*2]あるでゲソ。そう「継続 (Continuation)」でゲソね。というわけで、今回は「継続」の話をしておこうと思うんでゲソ。

[*3]

4.1 「継続」について勉強しなイカ?

まず最初に「そもそも継続ってなんなのか」を簡単に説明しておくでゲソ。

といっても、こんな本を読んでいるくらいだから「継続」っていう言葉は既に聞いたことがあるんじゃなイカ? そんなよく訓練されたお主たちのために超スピードで説明してみるでゲソ!

(1) 継続って何かというと「その後に行われる計算処理全体」のことでゲソ。
(2) さらに、一部の言語ではこの「継続」を実行時に捕捉して使うことができるんでゲソ (一部の言語っていうのは、例えば Scheme とか Scheme とか Scheme とかでゲソ。後は Ruby とか Standard ML とかもそうでゲソ)。
(3) で、そういった言語では「捕捉した継続」を好きなときに「起動」できるんでゲソ。継続を起動すると、捕捉されていた「その後に行われる計算処理」が実行されるんでゲソ。

... うむ、正直、何がなにやらよく分からないでゲソね (汗。なんというか、「わけがわからないよ」とか「この概念は出来損ないだ。使えないよ」という雰囲気がプンプンするでゲソ。

じゃあ、これをちょっと別の言い方で説明してみるでゲソ。

(1) 継続って何かというと、「その時点でのスレッドの状態」のことでゲソ。
(2) 一部の言語には、「スレッドの状態のコピーを取る」機能が用意されているんでゲソ。(そしてこの「コピーを取る処理」のことを「継続の捕捉」と呼ぶんでゲソ)
(3) で、取得したコピーを使うと、スレッドの状態を「コピーを取ったときの状態」に変更することが

[*1] といっても、前回の話との繋がりは一切ないので今回から読んでもらって全く問題ないでゲソ。
[*2] もちろん正確にはもっと一杯あるでゲソ。なにしろ Scheme は全てが大事と言っても過言ではないでゲソ。
[*3] ところで、タイトルはちょっとニンジャ○レイヤーっぽいでゲソが、中身はまるっきり関係ないでゲソ。タイトルだけの完全な出落ちでゲソね。まぁ、バー○ンハウスにでも引っかかったと思って許して欲しいでゲソ。

第5章

愚者のコントロール

— @tanakh

5.1 I

「わたし、気になります！」

やれやれ、また始まった。「やらなくていいことなら、やらない。やらなければいけないことなら、ギリギリまでやらない」がモットーの俺にとって、これは悪い予兆だ。子供のような無邪気な眼差しで四路戸えるに見つめられると、そんな信条とは裏腹に、どういう訳だかいつも面倒事に巻き込まれてしまう。

「私気になりません」

僅かばかりの抵抗を試みるが、最後には今回もこの眼差しに絡め取られてしまうのだろう。まったく厄介な役回りを引き受けてしまったものだ。

「いつもの遅延評価かい、奉太郎。そんなに無碍にしなくてもいいじゃないか。この謎、僕も気になるしさ」

複文里志はいかにも楽しそうだ。やめてくれ、俺はお前とは違う。ただ平穏な学校生活を送れればそれでいい。取り柄も何もない、普通の高校生なんだ。

「おはよう。.....どうしたの、折具？」

部屋に入るなり川中摩耶花はそう言った。憂鬱な気分が入り口からも分かるほど顔に張り付いているのだろうか。だが、いよいよもって逃げられそうになくなってきた。このままはぐらかすのと、四路戸に答えを提示するのと、どちらが評価コストが低いのだろう。必ずしも正しい答えを提示する必要はない。ただ、彼女が納得する答えを導き出せばいいのだ。俺は覚悟を決めた。

「ああ、分かったよ.....」

5.2 II

英国はエディンバラに留学している姉貴の言付けで、それなりに伝統ある古典論理部に入部したのがこの春のことである。そんなことでもなければ部活などという酔狂は、俺には無縁だったはずなのだ。高校に入学したとはいえ、特に何もやりたいこともない故、何をするかもよく分からない古典論理部なるものに入部したのがやはり何の間違いだったのか。

廃部寸前の古典論理部に入部しようという珍妙な奴がもう一人。部室に充てがわれている特別棟四階の地学講義室に入ると、そこに四路戸はいた。「豪農」四路戸家の一人娘にして、お嬢様らしい整った容姿。そしてそれに似合わぬ、旺盛な好奇心。答えを求めて俺を見つめる真っ直ぐな目には、何か逆らえぬものを感じた。

曰く、彼女の叔父は 40 年前のλ文字山高校古典論理部に所属していたらしい。そこで何やらとても悲しい出来事が起こり、彼は退学になってしまった。幼い頃その話を聞いてとても悲しい思い出のある四路戸は、40 年前の真相を確かめ、叔父の無念を晴らしたい、とまあそんな理由で古典論理部に入部