目次

第 0 章　関数型イカ娘とは!　@xhl_kogitsune　1
0.1　関数型イカ娘アニメ紹介 2

第 1 章　β簡約！λカ娘　@xhl_kogitsune　3
1.1　λしなイカ? 3
1.2　SKI しなイカ? 6
1.3　有効活用しようじゃなイカ! 10

第 2 章　数をつくらなイカ？　@nushio　11

第 3 章　関数型イカガール　@tanakh　17
3.1　関数型イカガール 第一話 17
3.2　関数型イカガール 第二話 18
3.3　参考文献 27

第 4 章　加速しなイカ？　@nushio　29
4.1　夢のようなライブラリがあったじゃなイカ！ 29
4.2　GPU も、FPGA も、あるでゲソ！ 30
4.3　実アプリの性能と向き合わなイカ？ 32
4.4　そんなの、私が許さないでゲソ！ 33

第 5 章　OS を侵略しなイカ？　@master_q　35
5.1　HFuse でファイルシステムを侵略しなイカ？ 35
5.2　ヲレヲレファイルシステムを作ってみるでゲソ! 36
5.3　fuseGetFileStat はファイルサイズを有限時間で返答しなければイカん 38
5.4　ということで完成したでゲソ 38
5.5　この後は何を侵略するでゲソ？ 38

会員名簿じゃなイカ?　40

第0章

関数型イカ娘とは!

— @xhl_kogitsune

Q. 関数型イカ娘って何ですか?
A. いい質問ですね!
Q. 関数型イカ娘について教えてください
B. その願いは君にとって魂を差し出すに足るものかい？

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。

2010 年 10 月のイカ娘アニメ第 5 話の次回予告で、イカ娘がパソコンを使っているシーン[*1]が映ったでゲソが、そこから

スーパープログラマーイカ娘!!! お、おぢょうちゃん、Scheme とか、どうかなっ!?
http://twitter.com/xhl_kogitsune/status/28710865318, 2010/10/26 02:28

次回予告でイカ娘がパソコンに向かっているシーンから、イカ娘は超絶 Scheme プログラマーという電波まで受信した←今ここ。そんな彼女に私は C++ とかを布教すべきだろうか? でも彼女は Scheme で幸せそうだし... うーん。
http://twitter.com/xhl_kogitsune/status/28711063147, 2010/10/26 02:31

という電波を@xhl_kogitsune が受信したのが始まりでゲソ。

その後も、アニメを見ながら twitter 上で関数型イカ娘ネタが増えて行ったでゲソ。詳しくはイカの Togetter を見るといいでゲソ。

[*1] http://www.ika-musume.com/season_1/images/middle/story/vol05/img_02.jpg

Togetter: **関数型イカ娘**　`http://togetter.com/li/63957`
　Twitter 上での関連発言まとめでゲソ。「見たけど分からなかったでゲソ…」という声も聞こえてくるでゲソ。

2010 年 12 月には Functional Ikamusume Advent Calendar jp 2010 というイベントがオンラインで開催されたでゲソ。

Functional Ikamusume Advent Calendar jp 2010　`http://bit.ly/fSoVhy`
　持ち回りで 1 日 1 題関数型イカ娘ネタを書こうという会でゲソ。12 人も参加者が集まったでゲソ。

そして、ついに関数型イカ娘本を出す計画が (なぜか) 実現したのでゲソ! ←今ここ
さぁ、お前たち、この調子で手続き型を侵略するでゲソよ!

0.1　関数型イカ娘アニメ紹介

関数型イカ娘の活躍はアニメにも出てきたでゲソ。

学校に行かないイカ？ (第 5 話)　わたしが学校のパソコンを使って Scheme でゲームを作っている記念すべき回でゲソ。わたしはゲームで遊んでいたんじゃなくてデバッグをしていたでゲソ!

勉強しないイカ？ (第 6 話)　「数学と λ 計算に突出した才能を示すイカ娘。しかし、海の家れもんの住人たちには相手にされず、イカ娘本人もまだ自分の真の力を理解していなかった……。」わたしには実は数学の才能もあったことが明らかになる回でゲソ。原作で数学が得意なら関数型が得意でもいいじゃないカ!

研究しないイカ？ (第 7 話)　わたしが数学科や情報科学科を侵略して研究活動をする回でゲソ。そして、わたしは人類侵略のためについに λ プロセッサの開発に乗り出すのだった……でゲソ!

新能力じゃないイカ？ (第 8 話)　新能力に目覚めたでゲソ。手続き型の能力かと憶測を呼んだでゲソが、そんなことはなかったでゲソ。

ピンポンダッシュしないイカ？ (第 9 話)　ネットワークを侵略しないイカ! まずは ping からでゲソ!

`make` しないイカ？ (第 9 話)　わたしと一緒に OCamlMakefile でゲソ! 海の家れもんにやってきた客の `make` に興味津々のイカ娘。イカ娘「その `make` はどうするのでゲソ?」客「`./configure; make; make install`」と `make` のし方を教えてもらうイカ娘。`make` しないのかと尋ねるイカ娘に対して、渚は `ant` 派だから `make` はしないのだと答える。調子に乗って `make` していたイカ娘だが、最後は `make clean` できなくなって栄子に泣きつくのであった。

第1章

β簡約！λ力娘

— @xhl_kogitsune

お前たち! わたしと一緒に λ しなイカ!

今日は、λ 計算と SKI 計算という、わたしが好きな二つの関数型計算モデルを紹介するでゲソ! どちらも、構文が 3 行で説明できるすごく単純な作りでゲソが、チューリング完全でゲソ。つまり、大雑把に言ってコンピュータで計算できることは全て計算できる素晴らしい能力を持っているでゲソ! 手続き型を侵略するでゲソ!

1.1 λしなイカ？

まずは楽しい楽しい λ 計算と β 簡約について教えてやるでゲソ! λ は怖くないでゲソよ〜

1.1.1 λ式を書かなイカ?

お前たちも生き物であるからには、関数を書きたいという欲求に勝てないはずでゲソ。λ 式はそんなお前たちにぴったりの関数の表現方法でゲソ!

え、なぜ関数は C 言語とかでも書けるじゃなイカって? これだから人間は困るでゲソ。λ 式で書く方が断然エレガントでゲソ。関数というからには式で書けるのが本当じゃなイカ。数式には `goto` なんて論外でゲソ。

最初に下準備として**変数**を考えるでゲソ。変数 x はそのまま λ 式として成立するでゲソ。

$$x$$

最小の λ 式じゃなイカ。ちなみに変数の具体的な名前はどうでもいいでゲソ。普通は小文字一文字を使うことが多いでゲソ。

さて、いよいよ**関数**を書こうじゃなイカ。x を受け取って $x+1$ を返す関数を考えるでゲソ。普通の数式だとこんな感じでゲソ。

$$f(x) = x + 1$$

これは、λ 式ではこんな感じに書くでゲソ。

$$(\lambda x.(x + 1))$$

λ は「らむだ」と読むでゲソ。入(はいる)でも人(ひと)でもイ(い)でもん(ん)でもないでゲソ。私の触手のように曲線が美しい文字じゃなイカ。λ の直後に書いてある x が引数で、その後の部分が関数が返す値を表す式でゲソ。これが λ 式での関数の書き方でゲソ。一般的には、x を変数名、A を λ 式とすると

$$(\lambda x.A)$$

は x を受け取って A を返す関数をあらわす λ 式でゲソ。正確には λ **抽象**と呼ぶでゲソ。私も覚えたての頃は一晩中砂浜に λ 式で関数を書きまくったものでゲソ。

$x+1$ というのは厳密な λ 式ではないでゲソが、読みやすいし、私の好きな Scheme とかを使うとこういう式も書けるから、説明のためにこう書くことにするでゲソ。全部純粋な λ 式で書く方法は後で説明するでゲソ。

関数が書けたら**関数適用**したくなるじゃなイカ。当然、λ 式で関数適用も書けるでゲソ。さっき書いた関数を 3 に適用して $f(3)$ を計算するにはイカのように書くでゲソ。

$$((\lambda x.(x + 1))3)$$

A, B を λ 式とすると

$$(AB)$$

第2章

数をつくらなイカ？

— @nushio

読者は、この本のそこかしこに SKI が飾り付けられているのに気がついたんじゃなイカ？これらの SKI 式はメッセージをエンコードしているので、できるものなら解読してみるがいいでゲソ。え？手書きな訳がないじゃなイカ！私は 1 0 本の触手があるとはいえとことん lazy(褒め言葉) なプログラマなのでゲソ。なるべく簡潔に書ける言語を選んで文字数最短の SKI 式を探索させたに決まってるじゃなイカ！

こうなったら、この SKI 文字列を生成した楽しいコードを解説することで、読者の脳を関数型に侵略してやるでゲソ?！ lazy(褒め言葉) な私は重要なポイントの解説しかしなイカら、コードの全文を `http://www.paraiso-lang.org/ikmsm/` からダウンロードし、あわせて読むでゲソ！

まず、このプログラムにおける SKI 式はイカのデータ型で表現されているでゲソ。

```
data Expr = M | S | K | I | Ap Expr Expr
          | Succ | N Int
          | Bottom String
          | Church Integer
          | Literal String
             deriving (Eq, Ord, Show, Read)
```

- このうち `M`, `S`, `K`, `I` はコンビネータでゲソ。前にも言ったように本当は `S` と `K` だけで用は足りるのでゲソが、輝かしい地上の支配者である `IKMSM` の名をしるすために `I` と `M` もアルファベットに加えるのでゲソ！ `Ap` は関数適用を表すでゲソ。`Ap` と `S`, `K`, `I`(, `M`) は SKI 式の根幹でゲソ。
- つづいて、`N` は整数、`Succ` は整数に 1 を加える 1 引数関数でゲソ。チャーチ数に対して `Succ` と `(Num 0)` を引数に与えると、そのチャーチ数がなんの整数を表していたのか評価できるじゃなイカ？だからこれらは検算に使わせていただくでゲソ。
- `Bottom` は計算がエラーになったことを表すでゲソ。エラーの原因を示す文字列を引き連れているのは便利のためでゲソ。
- `Church n` は、チャーチ数 n を部分式として利用することを表すでゲソ。再帰的に部分式を利用していくうちに、可能な部分式の場合の数は指数関数的に膨らんでいくでゲソが、`Church n` があればそんな状況をも少ない資源で表現できるでゲソ。
- `Literal` は、好きな名前を持ち、動作は未定義のコンビネータでゲソ。足など飾りじゃなイカ？

これら SKI 式の `eval` は次のように書けるでゲソ。

第3章

関数型イカガール

— @tanakh

3.1 関数型イカガール 第一話

高校一年の夏。

日も暮れてがらんとした「海の家れもλ」、そのテーブルで僕はノートを広げ一人考えに耽っていた。昼間の賑わいが嘘のように寂しげな砂浜を眺めながら、イカ墨パスタを口に運び、合間合間にペンを走らせる。至福のひとときだ。

「ポイントフリーでゲソか？」

いつの間にか、僕のノートを覗き込んでいる小さな頭があった。この海の家でアルバイトをしているイカさん。白くて妙な形をした帽子をかぶり、そこから青く長い髪が覗いている。いや、髪なのだろうか。髪のようなそれは風もないのにゆらゆらと空中を揺らめいている。まるで意識を持って動いているような...。

「仮引数を一つずつ落として、ラムダ式に書き換えるでゲソ。それから flip を使って束縛変数を後ろに移動させて変数を削除する、覚えるまでもないじゃなイカ？」

いいんだよ、練習しているだけなんだから。僕のノートにはいくつかの有名関数の定義が書いてあって、それらをポイントフリーに書き換えようとしている最中だった。

イカさんは髪で僕のペンを奪い、さらさらとノートに式を書き込む。矢印で左右から関数を挟み込んだような式だ。

「ほら、これはなんでゲソ？」

Squish パターンだ、僕は心のなかで答えた。Squid に掛けているのだろうか。突っ込むべきか迷ったが、結局口には出さない。二匹のイカが関数を侵略しているようにも見えるその式を、じっと見つめていた。

「分からないでゲソか？ Squish パターンじゃなイカ」

そう言ってイカさんは少し体を起こす。微かにイカ臭い香りが漂った。

イカさんはしたり顔で話し始める。「Squish パターンは f >>= a . b . c =<< g の形のポイントフリー式でゲソ。二つのモナドの計算結果をひとつの関数でまとめるのに使えるでゲソ。二つのものを一つのモナドに変換する、合成すると言ってもいいでゲソ。複数のモナドを合成できるようになったとき、とても素敵なことが起こるでゲソ」

イカさんの声を聞きながら、僕は別のことを考えていた。イカの女の子。侵略者。その二つの姿が、一人の少女であると気づいたら、どんな素敵なことが起こるんだろう。

でも、もちろん僕は何も言わず、黙ってイカ墨パスタを食べていた。

3.2 関数型イカガール 第二話

夏休み。

いつものように僕は日の暮れた海の家れも λ のテーブルでノートを広げていた。ノートには一列に並んだ数字。そしてそれらを書き換える操作。

「配列操作でゲソか？」

いつものようにイカの少女がノートを覗き込んできた。彼女はイカさん、関数の海からの侵略者だそうだ。今は訳あってここ、関数型海の家れも λ でウェイトレスをしている。小柄な少女の頭にちょこんと乗った白い特徴的な帽子から伸びる十本の触手がせわしなくうごめいている。

僕は配列について考えていた。関数型言語における配列。参照透明な言語における immutable な配列。これをうまく扱う方法について考えあぐねていた。mutable な配列を使うという選択肢もあるにはあるが、関数型言語としては美しくない。immutable でかつ読み書きを定数時間で行うことのできるデータ構造は無いものか———。

「ふーん」イカさんは中空を見つめながら「大きく分けて 3 つの方法があるでゲソ」勿体ぶってそう言った。

3.2.1 永続データ構造

「まず一つ目は、永続データ構造[*1]による配列操作の実現でゲソ」

purity と immutable とくれば persistent data structure に行き着くのはある意味自然だ。しかしこれは効率が犠牲になる。最も自明なのは、単純なデータの列として配列を表現する、つまり単なる配列の immutable 版だ。でもこれは更新の効率が極端に悪い。更新するたびに全てをコピーして新しい配列を作らなければならない。

「更新操作をまとめて行えば、効率は上げられるでゲソ。あるいは」少女はいたずらっぽく言った「構築に対して読み込みが多ければ問題ないじゃなイカ」

例えば静的な検索インデックスのような、構築一回に対して大量のリードオンリークエリが生じるのであればこのような単純な手法でも十分実用になる。しかしそういうケース以外だと採用は難しい。

「一般のケースを考えると、バランス木[*2]、フィンガーツリー[*3]あたりでゲソか」

バランス木を配列として用いるのもある意味自明だ。計算量が読み書きともにワーストケース O(logn) で可能だし、何よりシンプルだ。だけど、メモリ使用量、実行時間ともに命令型言語の mutable な配列を用いた場合に比べて激しく劣る。フィンガーツリーは Haskell には Data.Sequence[*4]として存在するがこちらもそんなに速いわけではない。これらは永続データ構造なのだ。永続性が必要ない場合にこれらはオーバーヘッドにしかならない。

イカさんは「永続性を捨てればいいじゃなイカ」またもいたずらっぽくそう言った。

3.2.2 Linear Type

それはそうだろう。だけど純粋関数型言語でそれをどうやって実現するのか。

まずひとつはモナドを使う方法だ。配列操作を副作用とみなして、すべての操作を IO モナドに包む。これは Haskell で言う所の IOArray[*5]だ。

「STArray[*6]もあるでゲソ」

そうだ。副作用の及ぶ範囲を特定のメモリ操作に限定したモナド、ST モナド。これを使うと外

[*1] http://en.wikipedia.org/wiki/Persistent_data_structure
[*2] 子の高さに偏りのない木のこと
[*3] http://en.wikipedia.org/wiki/Finger_tree
[*4] http://hackage.haskell.org/package/containers
[*5] http://hackage.haskell.org/packages/archive/array/0.3.0.2/doc/html/Data-Array-IO.html
[*6] http://hackage.haskell.org/packages/archive/array/0.3.0.2/doc/html/Data-Array-ST.html

第4章

加速しなイカ？

— @nushio

4.1 夢のようなライブラリがあったじゃなイカ！

人類よ、よく聞け！フリーランチは終わったでゲソ！これからは誰もが~~ボッタクry~~良心的価格のランチを得るために並列プログラミングという重労働を強いられる時代でゲソ。CPU もシンディーちゃんにブルドーザーちゃんに、どんどんベクトル化／マルチコア化してきているし、GPU なら既に何万というスレッドを並列に動作させることができるでゲソ。さらに将来イカに異様なハードウェアが現れても、我々はプログラムを書いてイカざるを得ないでゲソ。ならばプログラムを操る程度の能力を持った言語が是非とも必要でゲソ。いまこそ Haskell の力を見せつけて、闇の言語どもを侵略しようじゃなイカ！

Haskell には Accelerate `http://www.cse.unsw.edu.au/~chak/project/accelerate/`なる高速配列演算ライブラリがあって GPU 計算ができるでゲソ。最新のコードは彼らのレポジトリ `https://github.com/mchakravarty/accelerate` からチェックアウトでゲソ。チェックアウトしたら、そのフォルダで

```
> cabal configure
> cabal install
> cabal haddock
```

とやるとインなんとかされるでゲソ。`cabal haddock` を行えば `dist/doc/`イカにドキュメントが生成されるのでゲソ。Haskell のライブラリはいつだって、初見は意味不明不明じゃなイカ？それでも型を合わせて動かしてみたり、Haddock のハイパーリンクでいつも同じ所に飛ばされたりしているうちに、おぼろげながらライブラリの構造が見えてくるでゲソ。Haddock はとても役に立つじゃなイカ！

いくつか注意点があるでゲソ。まず執筆時点で、Accelerate が依存している CUDA という Hackage が、CUDA3.2 までしか対応していないため、最新の CUDA4.0 を入れた環境には Accelerate がインストールできない模様でゲソ。おなじく執筆時点の情報でゲソが、Hackage にある Accelerate(0.8.1.0) は github の (0.9.0.0) と比べてだいぶ遅れているので注意が必要でゲソ。この解説は Accelerate の開発者らによる論文 *Accelerating Haskell Array Codes with Multicore GPUs*, Chakravarty et. al. (2011) を基にしているので、そちらも参照でゲソ。

さて、Accelerate は、ただの GPU 向けコードジェネレータではなく、あくまでも Haskell プログラムに GPU 計算の速度を兼ね備えさせたいという発想でゲソ。そのため GPU 計算をモナドなどとして表現**せず**、純粋な計算にしていることが大きな特徴でゲソ。IO にとらわれず、Haskell プログラムの任意の箇所から GPU 計算を呼び出せるのでゲソ。実行中 GPU 計算が要求されると、その場で CUDA プログラムを生成し、コンパイルし、自身にリンクすることで結果を得る。素晴らしい技術じゃなイカ！

第5章

OSを侵略しなイカ？

— @master_q

もうユーザ空間は参照透明な海で征服しつくしたでゲソ! 今度は libc や kernel を関数型で侵略する作戦を練るでゲソ!!!

5.1 HFuseでファイルシステムを侵略しなイカ？

HFuse <http://hackage.haskell.org/package/HFuse>

を使えば Linux 上でファイルシステムを Haskell で書けるでゲソ。

まずは使ってみるでゲソ。環境は 2011 年 6 月 30 日頃の Debian sid amd64 でゲソ。Debian sid はインストールするタイミングによって ghc と haskell-platform のバージョンに不一致があるので、Debian パッケージの ghc まわりでインストールエラーが起きたら気長に修正されるのを待つか古いパッケージを選択インストールするでゲソ。

```
$ uname -a
Linux casper 2.6.39-2-amd64 #1 SMP Tue Jul 5 02:51:22 UTC 2011 x86_64 GNU/Linux
$ sudo apt-get install fuse libfuse-dev haskell-platform
$ sudo adduser あなたのユーザ名 fuse
... 再ログインするでゲソ ...
$ cabal install HFuse
```

これで HFuse パッケージのインストールが完了したでゲソ。github にサンプルがある [*1] ので試しに使ってみるでゲソ。

```
$ git clone git://github.com/realdesktop/hfuse.git
$ cd hfuse/examples
$ make
$ mkdir mountdir
$ ./HelloFS mountdir
$ mount | tail -n 1
HelloFS on /home/hogehoge/src/hfuse/examples/mountdir type fuse.HelloFS
(rw,nosuid,nodev,user=kiwamu)
$ ls -l mountdir
合計 1
```

[*1] https://github.com/realdesktop/hfuse/blob/master/examples/HelloFS.hs