目次

第 0 章　関数型イカ娘とは! / この総集編について　@xhl_kogitsune　v

0.1　この総集編について v
0.2　関数型イカ娘とは v
0.3　関数型イカ娘アニメ紹介 vi

第 I 部　簡約! λ力娘 (2011 年夏)　1

第 1 章　β簡約！λ力娘　@xhl_kogitsune　3

1.1　λしなイカ ？ 3
1.2　SKI しなイカ? 5
1.3　有効活用しようじゃなイカ! 9

第 2 章　数をつくらなイカ？　@nushio　10

第 3 章　関数型イカガール　@tanakh　16

3.1　関数型イカガール第一話 16
3.2　関数型イカガール第二話 16
3.3　参考文献 24

第 4 章　加速しなイカ？　@nushio　26

4.1　夢のようなライブラリがあったじゃなイカ！ 26
4.2　GPU も、FPGA も、あるでゲソ！ 27
4.3　実アプリの性能と向き合わなイカ？ 28
4.4　そんなの、私が許さないでゲソ！ 29

第 5 章　OS を侵略しなイカ？　@master_q　32

5.1　HFuse でファイルシステムを侵略しなイカ？ 32
5.2　ヲレヲレファイルシステムを作ってみるでゲソ! 33
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 34
5.4　ということで完成したでゲソ 35
5.5　この後は何を侵略するでゲソ？ 35

第 II 部　簡約!? λ力娘 (二期) (2011 年冬)　37

第 6 章　クラスタリングしなイカ？　@dekosuke　38

6.1　アルゴリズムとライブラリを決めなイカ？ 38
6.2　CSV をパーズしなイカ？ 40
6.3　実際にやってみるでゲソ 40
6.4　まとめでゲソ 41
6.5　参考文献でゲソ 41
6.6　kmeans-vector の license でゲソ 42

第 7 章　Copilot への希望と絶望の相転移　@master_q　**43**
7.1　Copilot はイカが？ . . . 43
7.2　Copilot DSL と C 言語コードの融合でゲソ . . . 46
7.3　Fibonacci.hs を例にして Copilot DSL の書き方を伝授するでゲソ! . . . 47
7.4　Wii リモコンで遊ぶでゲソ . . . 48
7.5　libc のないところでも操縦できるでゲッソ! . . . 51
7.6　けっきょく使えそうなんでゲソ？ . . . 55

第 8 章　加速しなイカ？！　@nushio　**57**
8.1　これまでのあらすじ . . . 57
8.2　進化しなイカ？ . . . 57
8.3　生存戦略しなイカ？ . . . 59
8.4　種をまかなイカ？ . . . 63
8.5　統計しなイカ？ . . . 64
参考文献 . . . 65

第 9 章　Java VM 上で tail call しなイカ？　@yryozo, @xhl_kogitsune　**66**
9.1　Java VM と関数型言語は相性が悪いでゲソ？ . . . 66
9.2　Java VM 上の関数型言語実装を紹介するでゲソ! . . . 70
9.3　Java VM 自体も侵略するでゲソ . . . 75
参考文献 . . . 83

第 10 章　関数型イカガール第 3 話　@tanakh　**86**

第 11 章　映画けいさん！　@tanakh　**91**

第 III 部　簡約!? 入力娘 (算) (2012 年夏)　**103**

第 12 章　運物語　@dif_engine　**105**
12.1　プロローグ . . . 105
12.2　ラーン・ユー・ア・ハスケル・フォー・グレート・グッド . . . 106
12.3　海の家「れもん」にて . . . 121
12.4　死闘 . . . 126
12.5　エピローグ . . . 131

第 13 章　RTS 海溝二万マイル　@master_q　**133**
13.1　RTS は Haskell コードを実行する VM でゲソ! . . . 133
13.2　[表層] Haskell の世界 . . . 134
13.3　潜水艦に乗り込もうじゃなイカ! . . . 134
13.4　[中深層] C 言語の世界 . . . 135
13.5　[漸深層] 幽霊船 . . . 140
13.6　Haskell の関数はどんなメモリ配置？ . . . 141
13.7　[深海層] cmm 言語の世界 . . . 143
13.8　浮上でゲッソ . . . 151
13.9　謝辞 . . . 152
13.10　参考文献 . . . 152

第 14 章　評価　@xhl_kogitsune　**153**
14.1　準備しなイカ? . . . 155
14.2　評価順序を変えなイカ? . . . 156

14.3 グラフ簡約しなイカ? . . . 161
14.4 Fully Lazy グラフ簡約 . . . 165
14.5 Optimal Reduction . . . 167
14.6 まとめと参考文献 . . . 173

第 15 章 オール・アバウト・ケーゾク・イン・スキーム @yryozo **175**
15.1 「継続」について勉強しなイカ? . . . 175
15.2 継続って誰が考えたんでゲソね? . . . 180
15.3 なんで Scheme と継続が関係あるのか不思議じゃなイカ? . . . 187
15.4 そろそろ CPS についてひとこと言っておくでゲソ . . . 193
15.5 「ゲンテーケーゾク」って聞いたことはないでゲソ? . . . 197
15.6 どうして限定継続が考え出されたのか気にならなイカ? . . . 199
15.7 他の種類の限定継続も紹介しておくでゲソ . . . 204
15.8 おわりに . . . 207
参考文献 . . . 207

第 16 章 愚者のコントロール @tanakh **209**
16.1 I . . . 209
16.2 II . . . 209
16.3 III . . . 210
16.4 IV . . . 211
16.5 V . . . 214
16.6 VI . . . 216
16.7 VII . . . 216
16.8 VIII . . . 217
16.9 IX . . . 217
16.10 X . . . 218
16.11 XI . . . 221
16.12 XII . . . 223
16.13 XIII . . . 223
16.14 XIV . . . 223
16.15 XV . . . 225
16.16 Epilogue . . . 225

第 IV 部 簡約! 入力娘 (4) (2012 年冬) 229

第 17 章 インターフェース @xhl_kogitsune **230**
17.1 Kihime Side . . . 230
17.2 Koyone Side . . . 236

第 18 章 Lighter than Light @master_q **243**
18.1 ダイエットの前に身体測定 . . . 243
18.2 ダイエット指標 . . . 244
18.3 開発環境 “フィットネスジム” . . . 245
18.4 作戦 I. integer-gmp パッケージをダイエット . . . 246
18.5 作戦 B. base パッケージをダイエット . . . 246
18.6 作戦 P. ghc-prim パッケージをダイエット . . . 250
18.7 作戦 R. GHC RTS をダイエット . . . 251
18.8 体が羽のよう! . . . 256

18.9 ダイエットは不健康？ . . . 259
18.10 シェイプ DOWN ガール . . . 260
18.11 参考資料 . . . 260

第 19 章 類は友を呼ぶ？ @darcshaskellmaster 262
19.1 参考文献 . . . 265

第 20 章 OCaml で printf じゃなイカ? @xhl_kogitsune 266
20.1 OCaml で printf しなイカ? . . . 266
20.2 どうやって format6 型になるでゲソ? . . . 267
20.3 フォーマット文字列をつながなイカ? . . . 268
20.4 参考文献 . . . 270

第 21 章 Haskell でも printf じゃなイカ?! @nushio 271
21.1 printf を再現するでゲソ！ . . . 272
21.2 型を追うでゲソ！ . . . 272
21.3 可変個引数関数の正体でゲソ！ . . . 277
21.4 まとめでゲソ！ . . . 278

第 22 章 けいさん！ highschool @tanakh 279

第 V 部 簡約!? 入力娘 Go! (2013 年夏) 299

第 23 章 めたせぴ☆ふぁうんでーしょん @master_q 300
23.1 はじまり . . . 300
23.2 レストランにて . . . 300
23.3 砂の中 . . . 302
23.4 最初のスケッチ: POSIX からの脱出 . . . 304
23.5 冷たい壁 . . . 310
23.6 二度目のスケッチ: 省メモリ化の追求 . . . 311
23.7 あこがれ . . . 320
23.8 三度目のスケッチ: コンテキストを操る . . . 322
23.9 突然の連絡 . . . 335
23.10 これからのこと . . . 336
23.11 参考文献 . . . 336

第 24 章 jhc コピペ @master_q 337
24.1 jhc たんへの愛の歌 . . . 337
24.2 参考文献 . . . 338

第 25 章 侵略者と転校生とアイドルとイカが再帰を学ぶそうですよ！ @nushio 339
25.1 侵略者 . . . 340
25.2 ネオオオサカ炎上 . . . 340
25.3 預言の書 . . . 347
25.4 強敵・災鬼獣 . . . 348

第 26 章 殺物語 @dif_engine 359
26.1 プロローグ . . . 359
26.2 準備：集合と関数 . . . 362
26.3 圏の導入 . . . 368
26.4 関手 . . . 374

26.5 自然変換 . . . 379
26.6 計算の抽象化と修飾された型 . . . 381
26.7 カンザスでないどこか . . . 390
26.8 もう一つの世界 . . . 394
26.9 死闘 . . . 398
26.10 エピローグ . . . 401

第 27 章 入力娘探索 ? @ark_golgo **404**
27.1 プロローグ . . . 404
27.2 囲碁のルールおさらい . . . 405
27.3 そもそもゲームにおける探索とは何か . . . 406
27.4 囲碁で探索が難しい理由 . . . 406
27.5 Df-pn . . . 407
27.6 λ探索 . . . 408
27.7 Df-pn λ探索 . . . 410
27.8 実験結果 . . . 411
27.9 エピローグ . . . 411
27.10 参考文献 . . . 411

第 28 章 ロマンティック・パージング @xhl_kogitsune **412**
28.1 ことのはじまり . . . 412
28.2 構文解析と文脈自由言語 . . . 413
28.3 Bottom-Up 構文解析 . . . 415
28.4 LR 構文解析 . . . 418

第 29 章 HaskEll Shaddai @fumieval **430**
29.1 Lens . . . 433
29.2 Traversal . . . 436
29.3 Iso と Prism . . . 437
29.4 Real World Lens . . . 439
29.5 リンク . . . 440

第 VI 部 簡約!? 入力娘 Rock! (2013 年冬) 441

第 30 章 入力娘探索 2? @ark_golgo **442**
30.1 プロローグ . . . 442
30.2 囲碁のルールおさらい . . . 442
30.3 AND-OR 探索 . . . 444
30.4 Min-Max 探索 . . . 446
30.5 α-β 探索 . . . 448
30.6 α-β 探索が囲碁ではうまく機能しなかった理由 . . . 448
30.7 原始モンテカルロ法 . . . 451
30.8 モンテカルロ木探索 . . . 452
30.9 モンテカルロ木探索を用いた囲碁ソフトの現状 . . . 453
30.10 エピローグ . . . 455
30.11 参考文献 . . . 455

第 31 章 僕のカノジョはスナッチャー @master_q **456**
31.1 川のほとり . . . 456
31.2 カノジョのこと . . . 456

31.3 Snatch-driven development . . . 457
31.4 Android NDK とそのサンプルアプリケーション . . . 458
31.5 スナッチ 1: 何もしない Haskell プログラム . . . 461
31.6 スナッチ 2: 最初のターゲット . . . 465
31.7 スナッチ 3: Haskell コードの近傍をさらにスナッチ . . . 471
31.8 こっそりと驚きを届けたい . . . 475
31.9 スナッチ 4: OpenGL ES をたたく . . . 475
31.10 スナッチ完了 . . . 480
31.11 プレゼントを作ろう . . . 481
31.12 お互いを理解することは小さく傷付け合うこと . . . 481
31.13 それでも明日はやって来る . . . 481
31.14 ソースコードライセンス . . . 482
31.15 参考文献 . . . 482

第0章

関数型イカ娘とは! / この総集編について

— @xhl_kogitsune

0.1 この総集編について

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。

本文収録内容

1 巻:「簡約！ λカ娘」 / 2 巻:「簡約！？ λカ娘 (二期)」/ 3 巻:「簡約！？ λカ娘 (算)」/ 4 巻:「簡約！ λカ娘 (4)」/ 5 巻:「簡約！？ λカ娘 Go!」/ 6 巻:「簡約！？ λカ娘 Rock!」

巻頭カラー口絵収録内容

- 総集編表紙別バージョン (七臣ナオミ 様)
- 1 巻「簡約！ λカ娘」表紙 (尾住 様)
- 2 巻「簡約！？ λカ娘 (二期)」表紙 (@dif_engine)
- 3 巻:「簡約！？ λカ娘 (算)」表紙 (@dif_engine)
- 4 巻:「簡約！ λカ娘 (4)」表紙 (七臣ナオミ 様)
- 5 巻:「簡約！？ λカ娘 Go!」裏表紙 (@dif_engine)
- 5 巻:「簡約！？ λカ娘 Go!」表紙 (@dif_engine)
- 6 巻:「簡約！？ λカ娘 Rock!」表紙 (@dif_engine)

0.2 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. 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

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

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。

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!

0.3　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イカ!

研究しなイカ？ (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イカ？ (第 9 話)　ネットワークを侵略しなイカ! まずは ping からでゲソ!

make しなイカ？ (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

$$(AB)$$

と書くと、関数Aを引数Bに適用する関数適用$A(B)$を表すλ式になるでゲソ。これで関数適用し放題じゃなイカ!

えっ、これだと引数を1つしか取れないじゃなイカ、って? 心配ないでゲソ! 引数なんて1つあればなんだって出来るでゲソ! たとえば、$g(x,y) = x+y$みたいな2引数関数は、

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

みたいにλ抽象2回で書けて、

$$((gx)y)$$

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１０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。

```
eval :: Expr -> Expr
eval expr = case expr of
  (Ap(Ap(Ap S f)g)x) -> let fx = eval (Ap f x)
                            gx = eval (Ap g x)
                        in eval (Ap fx gx)
  (Ap (Ap K x) _) -> eval x
  (Ap I x) -> eval x
  (Ap M x) -> eval (Ap (S&!I&!I) x)
```

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

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！

~~変態~~紳士たる皆様は Accelerate の中身が知りたイカ？ Accelerate が生成するコードは次のようにして確認できるでゲソ。まず cabal インストール時に`-fpcache` オプションを指定することで Accelerate が生成した CUDA コードが保存されるようになるでゲソ。

```
> cabal configure -fpcache
```

デフォルトでは`$HOME/.cabal/share/accelerate-x.x.x.x/cache/`に連番ファイル名で保存されるようでゲソ。これではどのファイルがどれなのか分からないじゃなイカ！ そこで例えば、Accelerate のソースにイ

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
-r--r--r-- 1 kiwamu kiwamu 20 1970-01-01 09:00 hello
$ cat mountdir/hello
Hello World, HFuse!
$ fusermount -u mountdir
```

うん、うまく動いているでゲソ。

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

第6章

クラスタリングしなイカ？

— @dekosuke

最近、ついったーのタイムライン・・・ゲフンゲフン、美しい海を見ていると、○○クラスタという単語をよく聞くでゲソ。例えば「関数型クラスタ」というと、関数型言語が好きな人たちの集まりを指すでゲソ[*1]。一般に、クラスタリングとは集団を似た者同士の小さなグループに分けることを言うでゲソ。クラスタリングは、その集団の性質を知るということにつながるでゲソ。言い換えれば、クラスタリングしてしまえばその領地を侵略したも同然でゲソ！！

というわけで、クラスタリングをやってみなイカ？

6.1 アルゴリズムとライブラリを決めなイカ？

やることは決まったので、クラスタリングのライブラリを探すでゲソ。Hackage のパッケージリスト

```
http://hackage.haskell.org/packages/archive/pkg-list.html
```

を、”Clustering”で検索すればいいんじゃなイカ？ ６件ぐらいヒットするんじゃなイカ？

このうち、”hierarchical-clustering” (階層クラスタリング）系のライブラリは興味深いけど話が複雑になるので今回は避けるでゲソ。hsgsom (自己組織化マップ) は高精度のクラスタリング手法でゲソが、ライブラリ作者がやる気を出しすぎて扱いが大変なので避けるでゲソ。

そして、残った希望、いやライブラリは・・K-Means のライブラリでゲソ。K-Means はクラスタリングにおける一番シンプルなアルゴリズムでゲソ。ソートで言うとクイックソートみたいなアルゴリズムでゲソ[*2]。

というわけで、kmeans ライブラリをインストール・・・するのが許されたのは 2011 年 10 月頭まででゲソ。なぜベストを尽くさなイカ？ 2011 年 10 月に kmeans より効率的な kmeans-vector ライブラリが作られたでゲソ。[*3]。List を使っている kmeans ライブラリと違い、kmeans-vector ライブラリは Data.Vector[*4]を使っているので 4 倍高速[*5]でゲソ！！！

```
cabal install kmeans-vector
```

で、必要なものはインストールできたんじゃないイカ？ え、ライブラリの内容が分からないでゲソか？大丈夫、K-Means は簡単なライブラリなので、中身を全部読めるでゲソ。短いのでここに転載してもいいんじゃなイカ？[*6]

```
type Vec = V.Vector Double
data Cluster = Cluster {
  cid :: !Int,
  center :: !Vec
  }
```

[*1] もちろん私も関数型言語クラスタに含まれているでゲソ

[*2] K-Means に関しては計算量がデータに対して NP 困難なので、大規模なデータに対しては近似手法が取られるでゲソ。今回はそんなことは気にしないことにするでゲソ

[*3] ちなみに Hackage のパッケージリストには載ってないでゲソが、理由は筆者の知識では分からないでゲソ

[*4] vector はランダムアクセスが定数時間で行えるデータ構造でゲソ。数値計算の人が使うのはだいたいランダムアクセスでゲソ

[*5] kmeans-vector ライブラリ作者調べ

[*6] kmeans-vector のライセンスは BSD3 に準じるでゲソ。章末に記載するでゲソ。

第7章

Copilotへの希望と絶望の相転移

— @master_q

みんないつも参照透明な海を泳いでるでゲソ？ 海のめぐみは大きいでゲソ。素晴しいでゲソ。でもそのめぐみは大きな力によって保たれているでゲソ。例えば簡単な Haskell 製のプログラムを ghc でコンパイルしてみるでゲソ。

```
$ ldd HaskellApp
  linux-vdso.so.1 =>  (0x00007fff4ad7c000)
  libgmp.so.10 => /usr/lib/x86_64-linux-gnu/libgmp.so.10 (0x00007f250dd37000)
  libffi.so.5 => /usr/lib/x86_64-linux-gnu/libffi.so.5 (0x00007f250db2a000)
  libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f250d8a7000)
  librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f250d69f000)
  libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f250d49b000)
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f250d116000)
  libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f250cefa000)
  /lib64/ld-linux-x86-64.so.2 (0x00007f250dfd6000)
```

依存ライブラリが多いでゲソ、、、特に厳しいのが、libgmp,libc,libpthread あたりでゲソ。これらのライブラリに強く依存していてなかなかこれらのライブラリのない環境では参照透明な海のめぐみを受けられないでゲソ。。。

7.1 Copilot はイカが？

そんなことに悩んでいたおり

```
Copilot <http://leepike.github.com/Copilot/>
```

を使うと Haskell の DSL で書いたプログラムが C 言語に落ちるという情報を tanakh 殿から仕入れたでゲソ。 Copilot を使えば、Haskell 製プログラムから libgmp,libc,libpthread などへの依存をなくせて、左記ライブラリのない組み込み用途の CPU でも Haskell プログラミングできるんじゃなイカ？ ちょっと使ってみるでゲッソ! 使い方は Copilot tutorial [*1] が詳しいでゲソ。

Copilot のしくみを簡単に解説すると、Copilot はある種の DSL で、

1. Copilot 外部の変数や関数を呼び出して、Stream という型に収め
2. Stream をフィルタしたり Stream 同士を演算したりして
3. 条件によって外部へ副作用を与える (trigger)

ような繰り返しを表現するでゲソ。このようにして書かれた Copilot DSL は C 言語に変換できるでゲソ。この Copilot の繰り返しを進めるために、Copilot DSL 由来の C 言語とは別に main() 関数を含む C 言語ソースを準備して、その main() 関数内で Copilot の step() 関数を繰り返し呼び出すようにするでゲソ。変換された C 言語はせいぜい libm ぐらいにしか依存していない [*2]ので、組み込み環境でも容易使えるでゲソ。

Copilot のインストールは簡単でゲソ。 Debian sid amd64(2011/12/03 時点) では、以下のような感じでゲソ。

[*1] https://github.com/niswegmann/copilot-discussion/blob/master/tutorial/copilot_tutorial.pdf

[*2] Copilot DSL 側で sqrt 関数などを使うと C 言語側の sqrt() 関数が必要になるでゲソ

第8章

加速しなイカ？！

— @nushio

8.1 これまでのあらすじ

■夢のようなライブラリがあったじゃなイカ！ これからは並列計算の時代だと聞いて闇の言語侵略に闘志を燃やす関数型イカ娘。そこにキュゥべえアイコンの人が現れて、避けようのない綻びも驚きもすべて君が抽象化してしまえば良い、そのための力が Haskell には備わっているし、彼女も出来ると説くのだった。

■GPU も、FPGA も、あるでゲソ！ イカ娘は Accelerate をつかった GPGPU コード生成を試してみる。Accelerate ちゃんは、これからの並列計算のコードを生成しまくっちゃいますからねーと意気込むのであった。

■実アプリの性能と向きあわなイカ？ 緑色っぽい CUDA 使いに性能の差を見せつけられた Accelerate ちゃんは、存在意義を見失い、その魂を黒く黒く染め上げていくのだった。

■そんなの、私が許さないでゲソ！ イカ娘は涙ながらに、自分自身の手でちゃんと速度の出るコードを生成してみせることを誓うのだった。いいよ、Accelerate ちゃんのためなら、私・・・。

■最後に残った半蔵門線でゲソ！ ついに東京をワルプルギスの夜が襲う。台風により交通網が麻痺する中、イカ娘たちは ICFP 会場から PFI 社までサイモン先生を護送する必要に迫られていた。

— そして、現状はというと・・・

8.2 進化しなイカ？

「フゥーハハハハハハハ！！ 数々の未来ガジェットを生み出してきた我がラボの最新作がついに完成したでゲソ！」

— イカちゃんは時間遡行者の真似をしている。個人的には、そっちじゃないほうの時間遡行者が好みなのだが。

「ぼやいたって仕方がないでゲソ。さあ、本題に行かなイカ？ この章では、拙作の偏微分方程式の陽的解法コードを GPU その他並列計算機向けに生成するためのドメイン特化言語、Paraiso(`http://paraiso-lang.org/wiki/`) を扱うでゲソ。Paraiso の実装の詳細については、函数プログラミングの集い 2011 などでも紹介したから説明を省略するとして、今回はそこからのアップデートをご紹介するでゲソ。なんと、Paraiso には遺伝的アルゴリズムベースの自動チューニング機構が備わったのでゲソ。すごいじゃなイカ！ これで、すべての偏微分方程式の陽解法を速くできるでゲソ。過去と未来、すべてのアーキテクチャで、すべてのコードを、この手で！」

— と言いたいところだが、今のところ対応しているのは OpenMP もしくは CUDA のみなのは相変わらずだ。それにしても、Paraiso の詳細を完全に飛ばしたら、さすがに以後の説明に差し障るのじゃなイカ。三行くらいでまとめられなイカ？

「だるいでゲソ！ それに三行じゃとても足りないでゲソ！」

「私からもお願いできるかしら」

「Orthotope Machine は、任意のサイズの配列変数を確保し、並列に操作する命令を持った、偏微分方程式の陽解法を表現するための仮想マシンです。」

第9章

Java VM上で tail call しなイカ？

— @yryozo, @xhl_kogitsune

お前たち、海の中でも海の家でも陸の上でも、Linux の上でも FreeBSD の上でも AIX の上でも x86 の上でも Power の上ででも、できることならいつでもどこででも関数型言語を使いたいと思うはずでゲソ。でも、言語処理系をマルチプラットフォーム対応にするのは大変な労力でゲソ。そこで、Java VM 上で関数型言語を実行したり実装したりしなイカ!?

9.1 Java VM と関数型言語は相性が悪いでゲソ？

最近は Java VM の上でも関数型言語を走らせている人が多くなってきているでゲソ。特に Scala や Clojure なんかが人気が高いようでゲソね。それもこれも私の侵略計画が順調に進んでいる証拠じゃなイカ？

Java VM の上での実装には、おおまかに 2 つの方法があるでゲソ。

インタプリタ　Java VM 上で実装したい言語のインタプリタを実行する方式でゲソ。

トランスレータ　実装したい言語のソースコードを、Java VM 上のプログラム (Java バイトコード) に変換し、その変換した Java バイトコードを Java VM 上で直接実行する方式でゲソ。実装したい言語の関数が、Java のメソッドに変換されるものが多いでゲソ。

ところが、Java VM と関数型言語 (のトランスレータ方式の実装) は相性があまりよくないという人たちもいるのでゲソ。というのも、関数型言語に必須と言ってもいい末尾呼び出しの最適化 (Tail Call Optimization, TCO) が Java VM ではすごくやりにくいのでゲソ。理由は下の表に書いた通りでゲソが、Java VM にはスタックフレームを積んだりポップしたりせずにメソッド外に飛ぶという命令がないのでゲソ[*1]。TCO がないと、Scheme で末尾再帰や tail call でループ書いただけで即死でゲソ!

分類	命令名
メソッド外に飛べるが、スタックフレームを積むかポップしてしまう	`invokevirtual`, `invokestatic`, `invokeinterface`, `invokespecial`, `return`, `areturn`, `ireturn`, `lreturn`, `freturn`, `dreturn`, `athrow`
スタックフレームの深さは変えないが、メソッド外には飛べない	`if_acmp`, `if_icmp`, `if`, `ifnonull`, `ifnull`, `tableswitch`, `lookupswitch`, `goto`, `goto_w`, `jsr`, `jsr_w`, `ret`

実際、Scala や Clojure の言語仕様を読んで見ると末尾呼び出しについては少し (というかかなり) 歯切れが悪いでゲソ...。とはいえ、関数型言語はこんな程度の問題ではびくともしないでゲソ。ちゃんと解決策はあるでゲソよ。

[*1] ちなみに、この問題は Java VM だけのものではないんでゲソ。例えば関数型の言語を C にトランスレートして gcc なんかでコンパイルしようとする時にも同じ問題が起こるでゲソ。

第10章

関数型イカガール第3話

— @tanakh

この作品はフィクションであり、実在する人物、地名、団体とは一切関係ありません。

---

12 月。冬の海に僕はいた。夏の賑わいは見る影もなく、寂しい海にはもちろん他には誰もいない。海の家は夏の終わりと共に閉店し、あのイカの少女も姿を消した。

それでも僕は週末が来るたびここに足を運ぶ。あの時の胸の高鳴りを忘れることができなくてーー

最近の僕を悩ませているのは、圏論だ。正確には圏論そのものではなく、代数学とプログラミングの関わりについてだ。圏論のことを考えているときはとても楽しくて、それだけで時間を潰してしまうことも少なくないのだけども、プログラミングの時にどことなく圏論の影がちらつく Haskell は勉強を始めた当初は手こずったりしたものだった。それが Haskell、あるいは関数型パラダイムのの敷居を上げているのではないか、少なくとも心理的障壁になっているのではないかと思ってしまう。

「そんなの決まっているじゃないカ！」

突然の声に僕は顔を上げた。そこにはあのイカの少女のくりくりとした大きな目があった。彼女は夕日を背に、僕のノートを覗き込んでいた。いつからそこにいたのだろう。どうやら僕は考え事をすると周りが見えなくなるようだ。しかしそんなことよりも、イカ娘にまた会えたことが嬉しかった。

「それは Composability のためでゲソ」

そんな僕にはお構いなしにイカ娘は続ける。

「Composability はプログラムを自由に組み合わせられる性質でゲソ。プログラムの Composability は、安心してプログラムを書くためにとても重要なことでゲソ。基本的なことじゃないカ」

Composability という概念は聞いたことがないでもなかったが、それがそんなに重要なものだとは考えていなかったわけで。

「コンビネータ、って聞いたことないでゲソか？ 人間の世界では小学校で習うって聞いたでゲソが」

はて、どこの世界の小学校の話だろうか？

「たけるがドラゴンの表紙の分厚い本を教科書に使ってるでゲソ。確か、ドラゴンなんとか[*1]とか...」

どういう小学校[*2]なんだ。それはともかく、コンビネータっていうのは自由変数を含まないラムダ式のことだ。狭義には関数を引数に取って、それを組み合わせて新しい関数を返すような高階関数を指すことが多いように思う。パーザコンビネータ[*3]などは典型的な例で、組み合わせ可能な小さなパーザから目的の関数を組み立てるというものだ。

「それで、Composability というわけでゲソ」

[*1] 「コンパイラ—原理・技法・ツール」通称 “ドラゴンブック”

[*2] Functional Ikamusume Advent Calendar 2011 一日目「たけるの通う小学校に Dragon Book や SystemC 等の専門書籍がてんこもりでヤバイでゲソ!」`http://d.hatena.ne.jp/xiaohuli3/20111201#1322758513` 参照

[*3] Haskell では *Parsec* `http://hackage.haskell.org/package/parsec` 、atto-parsec* `http://hackage.haskell.org/package/attoparsec` などが有名

第11章

映画けいさん！

— @tanakh

この作品は某映画を見たいけどなかなか見に行けない私の妄想の産物、つまりフィクションであり、実在する人物、地名、団体とは一切関係ありません。

「唯先輩、起きてください」
「むにゃむにゃ、あと 1 時間だけ……」
「ダメです。そんな時間ありませんよ」
「むにゃむにゃ、ハスにゃん。もう食べられないよ」
「何寝ぼけてるんですか。着きましたよ、ロンドン」
「……あ、おはよう、ハスにゃん」

私たちは今、ラムダ高計算機プログラミング競技研究部、通称「計算部」の課外活動としてイギリスに来ています。

飛行機の中でずっと私にもたれかかって寝ていたのが、平無駄唯先輩。いつもはとぼけているように見えて、今私たちがここに来れているのも唯先輩のおかげだったりします。

「ロンドンだー、ロンドンだー」
「ヒースローだー！」

あそこで唯先輩と一緒にはしゃいでいるのが、排中律先輩。2 年前に潰れかけだった計算部を復活させた人で、計算部の部長です。普段はいい加減な人ですが、いざというときは…やっぱり頼りないかも。

「おい、澪。何してんだ。早く荷物取ってこいよ」
「ううー…飛行機怖い……」
「飛行機はもう降りただろ。おい、しっかりしろー」

律先輩の隣で青い顔をしているのが、入文字山澪先輩。計算部で唯一の常識人で、実力もトップクラス。私のあこがれの先輩です。怖がりなのが玉に瑕かな。

「あらあら、ハスさちゃんは常識人じゃないのかしら？」
「あっ、モナ先輩。なんで私の考えてることを！？」

そしてこれが、型吹モナド先輩。おっとりしていて、すごいお嬢様なんだけど、ちょっと何を考えているのかわからない所があったりします。

「はあ、こんなので大丈夫なのかなあ」

海外に来ても、計算部は相変わらずです。

「ハスにゃん、すごいよ！ 白人がいっぱいいるよ！」
「もう、唯先輩。ここイギリスなんですから、当たり前じゃないですか。って、どこ行ってるんですか！？」
「えっ、こっちじゃないの？」
「乗り継ぎですよ！ ヒースローで降りちゃダメです」

はあ、こんなので大丈夫なのかなあ。

第12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。

12.1 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。

とにかく、僕達の会話はそんな風に始まった。

ありふれた出会い。

そしてありふれた会話。

しかし今にして思えばあの瞬間から——謎めいた暗殺者との死闘とその奇妙な結末へ至る運命のレールが敷かれてしまったのではないか——そんな風にも思うのだ。あの死闘の結末を思い出さずにはいられない。あの暗殺者は——。

第13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。

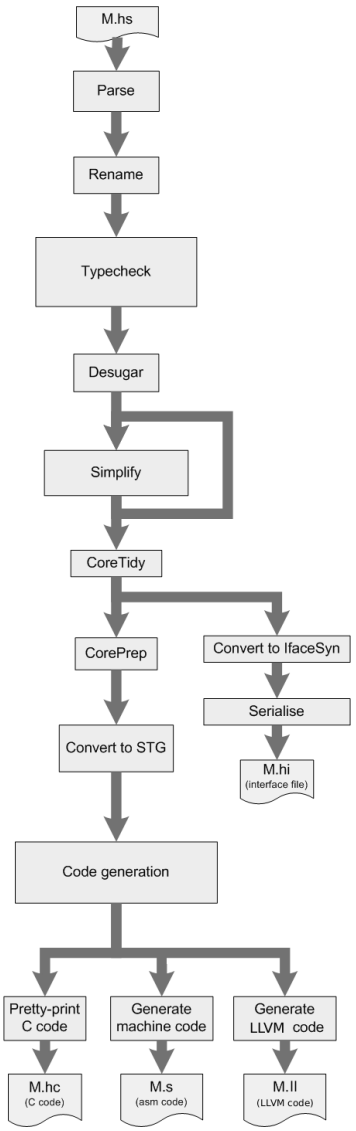
ところで、こんな楽しい世界 (GHC) がどんなしくみで動いているのか知りたくなイカ？

13.1 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オブジェクトファイルを作り、それらだけをリンクすれば実行バイナリになるのカ、というとそんなことはないのでゲソ。以下のようなプリミティブは (少なくとも現時点では)Haskell で書くことができていないでゲソ。



- コンパイル済み Haskell 関数の実行
- forkIO などで作った Haskell スレッドのスジューリング
- 並列実行

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

第14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?

さて、実は式の評価の順番 (評価順序) には色々あるでゲソ。この式を、どこから評価するでゲソか? C++ や OCaml なら、まず `1 + 2` を計算して `3` を出すところからはじめるんじゃなイカ? 然る後に関数 `(\x -> x * x)`[*1] を呼び出し、`x = 3` として `x * x` → `3 * 3` → `9` という風になるんじゃなイカ? これを式変形の過程として書くと図 14.1 のようになるでゲソ。

[*1] OCaml 版なら `(fun x -> x * x)`、C++ 版なら `f`

第15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]

15.1 「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」に変更することができるんでゲソ。(そしてこの「状態を変更する処理」のことを「継続の起動」と呼ぶんでゲソ)

... え、さっきと言ってることが違い過ぎなイカって? うーん、確かに一見すると全然違う話のように見えるでゲソね。だけど言ってることはほとんど同じなんでゲソ。

さっきの話だと、継続というのは「今後行う計算処理」だったでゲソ。だから継続を捕捉するというのは

[*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16章

愚者のコントロール

— @tanakh

16.1 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……」

16.2 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したとのことだ。その謎を俺がたまたま解いてしまったもんだから、事あるごとに気になることを付きつけられることになるわけだが、その話は又の機会に譲るとしよう。

そもそも古典論理部とは何をする部なのか？ そもそもの話として、古典論理とはなんなのか。実のところ俺にもよく分かっていない。姉貴に言われるままに入部しただけなのだ。部長を務める四路戸にもそこのところが分かっているのかは疑問である。俺達は未だに部活動らしきものをしたことがない。今のところ判明しているのは、文化祭で伝統ある文集「評価」を出すことだけだ。一体、何のためにある部活なのだろうか。

第17章

インターフェース

— @xhl_kogitsune

※本章は、某 LLVM 狐本[*1](の表紙) に端を発した非公式妄想です。妄想しか含まれていませんのでご注意ください。また、実在の人物 (勿論きつねさん含む)・言語処理系等とは関係ありません。

17.1 Kihime Side

「わたし」と「あなた」は、「インターフェース」で隔てられている。
「わたし」と「あなた」のカンケイは、「インターフェース」で切られている。
「わたし」は「あなた」とでなくても、同じようにやっていける。「あなた」にとって「わたし」は交換可能なものに過ぎない。
それが、「インターフェース」の持つ意味だから。
「インターフェース」の両側にいるのが誰だって、変わらないんだ。
—— キヒメ

「わたし」と「あなた」は、「インターフェース」でつながっている。
この「インターフェース」は、みんな同じ、みんな持ってる、誰とでも話せるものだけど。
「インターフェース」を流れる言葉は、ありきたりのものだけど。
「わたし」は、「インターフェース」を通して、「あなた」に話しかける。
「インターフェース」を超えて、「わたし」の心が伝わるといいなぁ、と思いながら。
—— コヨネ

わたしがコヨネと出会ったのはいつだっただろうか。最初は、どこにでもいるフロントエンドとしか思わなかったのだろう。フロントエンドなんて交換可能な短い付き合いなんだから、気にも留めなくて、覚えてもいない。これは、そんなわたしと、コヨネとのおはなし。

——

むかしむかし、コンパイラはひとつだった。原初の混沌のごとく、渾然一体としていて、ひとつの高級言語から、ひとつの機械語を生み出す、一枚板のナニカ。セカイはとても複雑に絡みあってはいたけど、とても単純だった。だって、そこには自分しかいないんだから。
時が下り、コンパイラはふたつに分かれた。天と地。フロントエンドとバックエンド。その両者をへだてる「中空」たる、インターフェース、中間表現。

> —— n 個の入力言語と、m 個の出力機械語があるとします。一枚板のコンパイラでこの全てをサポートするには、$n \times m$ 個の実装が必要です。でも、中間表現を間に入れれば、n 個のフロントエンドと m 個のバックエンドの計 $n+m$ 個の実装があればよいのです。

それはその通りだ。でも、わたしたちはフロントエンドとバックエンドに別れ、わたしたちは中間表現で隔てられてしまった。わたしたちの声は、中間表現という枠にはめられて、枠に入らなかったものは相手には届かない。枠にはまった声は誰のものでも同じで、わたしと、おなじものだったはずの相手とは天と地のように遠い。わたしは、誰が作ったのとも知れない中間表現を淡々と処理するだけの日常を送っていた。

[*1] 柏木餅子, 風薬 (著), 矢上栄一 (表紙): **3 日で出来る LLVM**, MotiPizza (2012).
http://d.hatena.ne.jp/motipizza/20120724 http://d.hatena.ne.jp/sabottenda/20120728

第18章

Lighter than Light

— @master_q

西暦 2005 年。覚えてイるカ？ 当時日本という国は ELF バイナリに熱狂していたでゲソ。Binary2.0 カンファレンス [*1] の開催。書籍 Binary Hacks [*2] の出版。その Binary Hacks の中に「25. glibc を使わないで Hello World を書く」という章があったことを覚えている者もいると思うでゲソ。それから「A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux」 [*3] という記事が Web に公開されたこともあったでゲソ。なにもかも懐しいでゲソ!

これらの試みは「ELF 実行バイナリのファイルサイズを小さくする」ことを目的としていたでゲソ。そこで、今回は GHC が Haskell コードをコンパイルして吐き出す ELF 実行バイナリをどこまでダイエットできるのか挑戦してみるでゲッソ! 今回対象とする実行環境は Debian GNU/Linux amd64 sid 2012/12/01 時点、他詳細はイカでゲソ。

```
$ gcc --version | head -1
gcc (Debian 4.7.2-4) 4.7.2
$ /usr/local/ghc7.6.1/bin/ghc --version
The Glorious Glasgow Haskell Compilation System, version 7.6.1
```

18.1 ダイエットの前に身体測定

ダイエットの前には身体測定でゲソ。まず削減対象となる Haskell コードを作ってみなイカ？

```
-- File: Fib.hs
module Fib where
import Foreign.C.Types
foreign export ccall fib :: CInt -> IO CInt

fibonacci :: [CInt]
fibonacci = 1:1:zipWith (+) fibonacci (tail fibonacci)

fib :: CInt -> IO CInt
fib n | 0 <= n && n <= 40 = return $ fibonacci !! fromIntegral n
      | otherwise = return 0
```

```
/* File: CMain.c */
#include <stdio.h>
```

[*1] http://0xcc.net/blog/archives/000078.html http://0xcc.net/blog/archives/000149.html
[*2] http://www.oreilly.co.jp/books/4873112885/
[*3] http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html

第19章

類は友を呼ぶ？

— @darcshaskellmaster

いいか、あまり説明している時間がないからよく聞いてくれ。この音声を同志諸君らが聞いているということは、私はもはや・・・まあいい。我々型レベル Haskeller はこれまで大変な不自由を耐えてきた。が、それも 7.4.2 までの事・・・遂に決起の時は来たのだ。型レベル整数及びその演算子の実装である！ 仮令私が斃れても選ばれれた同志たる諸君らが必[redacted]ヤの支配から人民を解放せねばならぬ！ 残念ながら現時点では、型レベル整数の演算の評価は開発ブランチ留りだ。同志諸君らの時代にはこれがメイン入りして居ることを確信しつつも、未然の備えとして `type-nats` ブランチの導入法のメモを残そう：

```
$ git clone git@github.com:ghc/ghc.git
$ cd ghc/
$ git checkout -b type-nats origin/type-nats
$ time ./sync-all get
(4m27.533s)
$ cp mk/build.mk.sample mk/build.mk
$ vi mk/build.mk
$ diff mk/build.mk.sample mk/build.mk
17c17
< #BuildFlavour = quick
---
> BuildFlavour = quick
177d176
<
$ ./boot
(12.231s)
$ ./configure
$ make -j4
(11m52.788s)
$ make binary-dist
(22.616s)
$ ls *.tar.bz2
ghc-7.7.20121111-x86_64-unknown-linux.tar.bz2
```

こうして「梱包物」を製作したら、 hsenv [*1]を用いて敵に浸透の事実を気づかれぬよう環境を導入する。

```
$ hsenv --ghc=src/ghc/ghc-7.7.20121111-x86_64-unknown-linux.tar.bz2 --name=7.7.
20121111
```

[*1] https://github.com/Paczesiowa/hsenv

第20章

OCaml で printf じゃなイカ?

— @xhl_kogitsune

20.1 OCaml で printf しなイカ?

お前たち、C や C++ で printf する時、表示したい値の型とフォーマット文字列を間違えて残念な気持ちになったことはなイカ?

```
printf("%d: %s\n", "hoge", 3); // まちがい! 何が表示されるかわからないでゲソ!
```

もちろん、gcc -Wall オプションなどを使えば、コンパイラが解析してイカのような warning を出してくれるでゲソが、ちょっと頼りないでゲソ[1]。

```
warning: format '%d' expects type 'int', but argument 2 has type 'const char*'
warning: format '%s' expects type 'char*', but argument 3 has type 'int'
```

OCaml の Printf.printf (及び Printf.fprintf などの類似関数) なら、ちゃんと表示する値の型がコンパイラにチェックされるでゲソ! 素晴らしいじゃなイカ!!

イカではコマンドプロンプトから ocaml と打って対話モードでプログラムを入力・実行しながら解説するでゲソ。(* と *) で囲まれている部分は説明のために追加したコメントでゲソ。

```
# Printf.printf "%d\n" 1;;          (* こんな風に書くと 1 が表示されるでゲソ! *)
1
- : unit = ()
# Printf.printf "Hello, %s!\n" "world";; (* Hello, world! が表示されるでゲソ! *)
Hello, world!
- : unit = ()
# Printf.printf "%s\n" 1;; (* コンパイル時に型エラーになるでゲソ! *)
Error: This expression has type int but an expression was expected of type
         string
```

printf とフォーマット文字列だけ取り出して型を見ると、イカのようにちゃんと型がついているでゲソ。

```
# Printf.printf "%d\n";;
- : int -> unit = <fun>    (* %d なので引数は int    になっているでゲソ! *)
# Printf.printf "Hello, %s!\n";;
- : string -> unit = <fun> (* %s なので引数は string になっているでゲソ! *)
```

ここでの "%d\n" とかの文字列リテラルに見えるものは、実は string 型ではなくて 特殊なビルトイン型 (6 引数の多相 Pervasives.format6 型) なのでゲソ。型アノテーションを指定するとフォーマット文字列部分の型とかが見れるでゲソ。

---

[1] 感覚には個人差があります。gcc は warning は出してくれますが printf の各引数に静的型を割り当てるわけではないので、printf("%lld: %lld", 1, 2); のような場合でも暗黙の型変換はしてくれなかったり。まぁ OCaml はそもそも暗黙の型変換はしないのですが……。

第21章

Haskellでもprintfじゃなイカ?!

— @nushio

コンパイラがサポートする特殊なビルトイン型を使うとかあもりにもひきょう過ぎるでしょう？ 汚いなさすがニンジャ汚い！

「—などと思ってしまったヘッズはケジメでゲソ！ 海のように広い心を持って、様々なアプローチを受け入れようじゃなイカ！ 前章ではOCamlの`printf`に触れたので、お次はHaskellの`printf`を調べなイカ？ 今回はかの高名な光の言語使いたるダークハスケルマスター、の契約者である高階(たかしな)さんに来てもらったでゲソ！」

「むろん、ギーメル・ヘー・コフ派の固有術式である原典創作の秘跡をもってすれば、コンパイル時に日常会話の中にすら任意の教義を見出し、望みの型の関数に変換することなど余裕。だが、Haskell 98の範囲内においても、型クラスを活用することで、一にして多貌なるもの、一つの真名に様々なarity、様々な型の引数の関数の化身を備わらせることは可能。[*1]」

「なんかすごい技術でゲソ！ 覚えておけばどこかで役に立ちそうじゃなイカ？ Haskellの`Text.Printf.printf`はイカのように、いくつでも引数を渡せるし、結果を文字列としても`IO`モナドとしても使うことが出来る仕様でゲソ。」

```
import Text.Printf

main = do
  -- まずは IO a として使うでゲソ！
  printf "one over seven = %6.5f\n" (1/7 :: Double)
  printf "%d %d\n" (1:: Int)  (3:: Int)

  -- 次に文字列として使ってみるでゲソ！
  let filename :: String
      filename = printf "%s-%d.txt" "script" (5::Int)
  writeFile filename "hello world!\n"
```

「そこかしこに型注釈しないと動かないのはちょっと残念でゲソが、便利じゃなイカ！」

[*1] GHC拡張のTemplateHaskellを使えば通常の文字列をパースして任意のHaskellソースを生成できる。THを使った型安全なprintfの実装については`http://okmij.org/ftp/typed-formatting/`を参照。またHaskell98の範囲内で、一つの名前が可変個の引数、様々な型の引数を取る関数を表すようにできる

第22章

けいさん！　highschool

— @tanakh

～これまでのあらすじ
大好きな唯たちが卒業して一人残されてしまったあずにゃん...。けいさん部の部室は今日も一人ぼっち...可愛そうなあずにゃん。でも、そんな落ち込んでいるあずにゃんの親友である憂ともう一人の子は新けいさん部として立ち上がるべく仲間に！ その後も偽ムギちゃんやメガネも加わりいよいよ情報大航海へ！？ 一大スペクタクルが今、開幕！

http://www.mangaoh.co.jp/catalog/295626/ より

先輩たちが卒業してから、しばらくが経ちました。元々「けいさん部」もとい、計算機プログラミング部なんてところに入るとは、入学するまでは夢にも思っていなかったのだけれども、あれから二年も経つ今となっては、もうそこにいない私は考えられなくなっていて———。私は今もここにいる。でも、唯先輩たちは、もうここにいない。少し前まではここにいたはずの人がここにいない、それがこんなに寂しいものだなんて、考えていなかったんです。いや、そうじゃなくて。考えたくなかったから、考えないようにしていただけだったんだ。

「———二人きりになっちゃったね」

水槽をゆらゆらとたゆたうトンちゃんにひとりごちます。先輩たちが卒業して、下級生の部員もいなくて、それなりに広い部屋に、私はひとりぼっち。それを考えたくなくて、無意識にカメに話しかけてしまう。去年の今頃、私が二年に進級するとき。けいさん部に新入部員が入らなくて、代わりに新メンバーとしてスッポンモドキのトンちゃんを迎えたんだった。

「———大きく—なったね」

飼い始めた後でわかったことだけど、スッポンモドキは成長するとかなり大きくなるといいます。こんな小さな水槽ではじきに飼えなくなる。そうなったら、今度こそ本当にひとりぼっちになってしまう。

「ひとりに———しないで———」

私以外に誰もいないただひとりの空間にて、緊張を保つ理由はありませんでした。頬を伝う涙をそのままに、春の陽光射し込む音楽室、私の意識はぼんやりとしていました。何気なしに、幾度となく繰り返した、ついこの間までの楽しい思い出を蘇らせます。「あ、あずにゃん。こんなところにいたんだ。探したんだよー！」やめてください、唯先輩。頬を擦り付けないでください。ここ、人前ですよ。「だってあずにゃん可愛いんだもん。すりすり」やめてください、本当に怒りますよ。———やめないで！

体をピクリとさせながら微睡みから覚めると、どれぐらいの時間が過ぎたのか、誰かが部室の扉をノックする音の後、しばらくしてカチャリと扉が開きました。

「すいませーん、けいさん部の部室って———」

「———けいさん部の部室って、こちらでしょうか？」

ああ、開けてしまった。ついに開けてしまった。部活紹介のライブがかっこよくて、気になっていたのだ

第23章

めたせぴ☆ふぁうんでーしょん

— @master_q

23.1 はじまり

西暦 2013 年。凶の年。ソフトウェアエンジニアの多くは Web の世界に住んでいた。じゃうぁすくりぷと、あんどろいどじゃうぁ、おぶじぇくとしー、しーしゃーぷ。高機能な言語と環境をあやつり、彼等は栄華をきわめていた。きらびやかな UI、きらびやかな通信。またある人々はしーげんごとしーぷらすぷらすをあやつり、匠の技で複雑なハードウェアをあやつり、複雑多機能なうぇぶぶらうざを作り、そしてそれらを支えるオペレーティングシステムりなっくすを作っていた。

しかしぼくらは幸か不幸かそのどちらでもなかった。オープンソースのソフトウェアを使って組み込みデバイスを作っていたのだ。世間からは考えられないほど古いバージョンのオープンソースを使い、世間からは考えられないほど汚れたソースコードを弄り、そして世間からは考えらない奇妙なデバイスの集合を組み合わせていた。それだけならまだよいのだが悲しいことに作った製品への世間からの評価は低かった。ぼくらの中にはこの製品化の渦に飲み込まれて息絶える者もいた。

そうして生き延びたぼくたちは現実から逃避するようにインドネシアに逃げのびた。もう設計などしたくはなかったのだ。自然豊かなこの国で、ぼくたちは安らぎ、ソフトウェアのことなど忘れかけていた。それでも組み込みシステムへの想いは捨てきれず、何か改善する手はないかと失われたテクノロジ **型システム**を学び会得していた。しかし祖国を離れてまともな仕事がある訳もなく、先進国向けの Web サービスを作るアルバイトで食い繋いでいた。もはや組み込みの民はこのまま消えゆくしかないのか……そう思いかけたとき……

「ワシが **めたせぴ**☆ でゲソ!」

どこからともなく静かだが力強い声が響いた。

「これからおぬし達は一本の糸をたどることになるでゲソ。この糸は古えのテクノロジー型システムによって綿密に計算/予測された運命なのでゲソ!」

なにかわからないが、安心とやはり不安がいりまじって混乱する。この娘はいったい何を言っているのだ？

「今おぬし達を囲んでいる状況はよくわかっているでゲソ。増えるカスタムシステムコール。多種のデバイス。崩壊しつつある旧来の CPU アーキティクチャ。POSIX API 上に構築された複雑なミドルウェア。絶え間ない製品リリース。猛進進化する型付けされていないオープンソースソフトウェアへの追従」

この娘は何者だろう？ しかし、そうだ。その通りだ……

「そしておぬし達は今、型システムを手にしている。それがおぬし達だけが持つ特性、もしくは力じゃなイカ？ それならこれから取るべき道はあまりにもわかりきっているでゲソ!!!」

そう言うと、めたせぴ☆と名乗る娘は消えてしまった。……あれは夢だったのか？ 今となってはぼくたちにもわからない。

23.2 レストランにて

ぼくたちはちょっと贅沢をして海辺のレストランに来ていた。いつも作っている Web サービスではない別の話をしたくなったのだ。さっきの娘にそそのかされたわけじゃない、ほんの気晴らし。それでもぼくたちの話題はいつも組み込みシステムに向いた。あの頃ぼくたちを苦しめていた問題の根っこは何だったのだろう？ 製品開発において最も重要なのは不具合の解消だ。不具合の解消方法には根本対策と暫定対策がある。根本的に対策できれば 100 点満点だが、工数の関係上たいていの対策は暫定策に終わる。暫定対策を施して

第24章

jhcコピペ

— @master_q

24.1 jhcたんへの愛の歌

聞いてください！

日本語版

jhc！ jhc！ jhc！ jhcぅぅううわぁあああああああああああああああああああああああああああん！！！ あぁあああああ…ああ…あっあっっー！ あぁあああああああ！！ jhcjhcjhcぅううぁわぁあああああ！！！ あぁクンカクンカ！ クンカクンカ！ スーハースーハー！ スーハースーハー！ いい匂いだなぁ…くんくんんはぁっ！ jhcたんのオレンジの髪をクンカクンカしたいお！ クンカクンカ！ あぁあ！！ 間違えた！ モフモフしたいお！ モフモフ！ モフモフ！ 髪髪モフモフ！ カリカリモフモフ…きゅんきゅんきゅい！！ darcsのjhcたんかわいかったよぅ！！ あぁぁああ…あああ…あっあぁあああああ！！ ふぁぁああああんんっ！！ Parsec使えて良かったねjhcたん！ あぁあああ！ かわいい！ jhcたん！ かわいい！ あっああぁああ！

haskell2010化されつつあって嬉し…いやぁあああああああ！！！ にゃあああああああああん！！ ぎゃあああああああああ！！ ぐあああああああああ！！！ ソースコードなんて現実じゃない！！！！ あ…バイナリもよく考えたら…

j h c ちゃんは現実じゃない？ にゃあああああああああああああん！！ うぁあああああああああ！！ そんなぁあああああ！！ いやぁぁぁあああああああああ！！ はぁあああああああん！！ NetBSDぁあああ！！ この！ ちきしょー！ やめてやる！！ Haskellなんかやめ…て…え！？ 見…てる？ スレッド対応のjhcちゃんが僕を見てる？

STM搭載のjhcちゃんが僕を見てるぞ！ jhcちゃんが僕を見てるぞ！ 中断GC対応のjhcちゃんが僕を見てるぞ！！

コンパイラのjhcちゃんが僕に話しかけてるぞ！！！ よかった…世の中まだまだ捨てたモンじゃないんだねっ！ いやっほぉおおおお！！！ 僕にはjhcちゃんがいる！！ やったよJohn！ ひとりでできるもん！！！ あ、独自RTSのjhcちゃあああああああああああああん！！ いやぁあああああああああああああ！！！！ あっあんああっああんあgrin様ぁあ！！ リ、リージョン推論！！ superkindぉおおおおお！！！ superboxぅぅううう！！

ううっうぅうう！！ 俺の想いよjhcへ届け！！ サンタモニカのJohnへ届け！

第25章

侵略者と転校生とアイドルとイカが再帰を学ぶそうですよ！

— @nushio

要旨

最近レコードアクセスのために lens を使う機会が増えてきたので、ちゃんと勉強しようと思いたち、“Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire” を読むことにしました。そしたら、未定義・意味不明の記号があまりにたくさん使ってあってとても読めたものではなく、他の論文やブログ記事を参照し、ロゼッタストーン片手に古代文字を解読するような作業の結果、ようやく読めるありさまでした。先駆的論文としての価値は強調してしすぎることはありませんが、より新しい論文や、ライブラリで補完・改善されている点も見えてきました。そこで、この論文を読もうという人にも、そうでない人にも私の体験をたのしくお伝えすべく古文書の解読をめぐる冒険活劇に仕立てることにしました。本文中で「旧約甘蕉書」とされている文書のページ数や式番号は原論文のそれに準じています。原論文は Web からダウンロードできますので、印刷して片手に置きながらお読みください。また、本文に登場するソースコードは `https://github.com/nushio3/`で公開しますので、こちらもあわせてご覧ください。

ちなみに、甘蕉はバナナという意味です。

ちなみにちなみに、レコードアクセスのための lens と本稿で解説する lens は結局別物だったようです。ひどい話ですよね。本来知りたかった方の lens の記事は——たぶん明日の出来事だ。

序

「ブンブンブブーンブンブブーン、ブンブンブブーンブンブブーン、ブンブンブブーンブンブブーン」

少女はエンジン音の口真似をしながら薄明のヒガシ＝オオキイ＝メインストリートを駆けていく。と、体を傾け路地の一つに駆け込むや彼女の水色の髪の一部が触手めいて伸び、防水リュックから取り出した配達物を郵便受けに叩き込む。

「イヤーッ！」「イヤーッ！」「イヤーッ！」

叫び声の源は住宅街に紛れていった。キョート＝サン市街が活動を開始し、λ文字山から日が昇らんとしているのがわかる。ヒガシヤマ＝マウンテン連峰の向こう側、まだ暗い夜空に聳え立つ円柱状の構造物が水平線下からの太陽光を散乱して、いち早くキョート＝サン市街に光源を届けているからである。

これこそ、納紀元年に完成し、世界のなりたちを根本から変えることになった発明、余剰次元エンジンであった。

この物語はフィクションであり、冒頭から末尾まで、登場する人物・組織・団体・事象は実在および架空のそれらとは一切関係ありません。あなたがどこかで出会ったことがある誰かや何かが登場するかもしれませんが、それは他人の空似、同姓同名の別人、一卵性および二卵性双生児、生き別れの義理の妹、メイドインなんちゃらの模造品等でございます。あしからずご了承ください。

第26章

殺物語

— @dif_engine

物語の概要と目的

この章では、モナドについて数学的とプログラミングの関係を意識しながら説明します。例えば関数型プログラミング言語の一つである Haskell では、モナドは入出力を始めとする多くの標準的なライブラリで採用されています。モナドは圏論（けんろん）という抽象的な数学の枠組みで物事を捉えたときに認識された概念です。

単に幾つかのライブラリの使い方を習得したいだけならばサンプルコードの学習で十分であり、圏論の学習は必要ないのかもしれません。しかしながら、圏論におけるモナドの意味や、モナドとプログラミングとの関係を理解すれば、モナドというデザインの有効性と限界を理論的に把握することができるでしょう。
以下では Haskell をすでにある程度習得している読者を想定し、

- Haskell と圏がどのように関わっているか
- モナドはどのようなプログラミングのアイデアを抽象化しているか
- モナド則をどのように理解するか

という順序で解説します。本記事のストーリー部分は「簡約！？ λカ娘（算）」の「蓮物語（はすものがたり）」の続編になっていますが、数学やプログラミングについての内容に関する限りにおいては「蓮物語」を前提としておらず、独立した記事としてお読み頂けます。

26.1 プロローグ

26.1.1 モナドって何でそんなにすごいんですか？

９月になっても暑い日が続いていたが、秋の気配は深まっていた。
そんなある日のこと、私の部屋には忍野扇（おしのおうぎ）ちゃんがいた。
「――それで翼（つばさ）さん、Haskell のモナドって何でそんなにすごいんですか？」
ほぼ初対面に近い下級生が私の家を訪問――アポ無しで――してモナドについて質問をする。
なんだか奇妙な構図だった。
奇妙ということなら、そもそも扇ちゃん自身にも少し奇妙なところがある。
といっても奇妙なのは彼女の性格のことではなく――いや、性格も少し変わってるかもしれないが――一連の怪異がらみの事件との関係だ。

26.1.2 忍野扇

その日の朝、空にはうろこ雲が一面に広がっていた。
空の東の端を朝日が燃え立つようなオレンジ色に染め上げていた。空を埋め尽くすうろこ雲はオレンジから白へ、そして白から青灰色へと色を変えながら夜闇の名残をとどめる西の空に溶け込んでいる。整然と並ぶ一つ一つの雲の小片の東側は曙光を受けて金色に輝いていた。オレンジ、青灰色、濃紺によって染め分けら

第27章

λ力娘探索 ？

— @ark_golgo

27.1 プロローグ

図 27.1: λカ娘三連敗 ！

「あううう、また負けたでゲソ」
「これで儂の 3 連勝。やはり置石が足らんかな」
「そ、そんなことないでゲソ。4 子も置いて負けたのはたまたまでゲソ! 次は勝つでゲソ!!」
「まあその意気込みだけは買ってやるがな」
λカ娘は最近ある老人と囲碁を打つようになった (図 27.1)。囲碁に自信があったλカ娘であるが、この老人には 4 子のハンデ (段級位にして 4 段級差に相当) を付けてもらってもまるで勝てなかった。
「あううう、ああは言ったものの、そう簡単に勝てる相手ではないことは良くわかるでゲソ。囲碁はそう簡単に強くなれるゲームでは無いでゲソ。私も囲碁歴は 10 年近くて、アマ三段はあると思うでゲソが……。あの老人は何者でゲソか……。特に読みの力が全然違うでゲソ。うーん、仕方ないでゲソ。こうなったら……、自分で囲碁の探索プログラムを作ってそれを使ってカンニングするでゲソ! 」
そう考えたλカ娘は、早速図書館で囲碁の探索に関する論文を探し始めた。しばらくして、『証明数と反証数を用いたλ探索』という論文と、『分岐因子が一様な探索空間のための AND-OR 木探索アルゴリズム』という博士論文、そして『ゲーム計算メカニズム』という書籍を見つけた。
「よし、これを基に実装するでゲソ。でも博士論文はちょっと読みこむの大変そうでゲソ。あくまで参考にとどめて、一番目の論文を基に実装するでゲソ」
さて、λカ娘が取り組んでいる囲碁の探索について、ここでもλカ娘に紹介してもらうことにしよう。なお、すでに分かっている読者も多いと思うが、今回の話は関数型とは関係なく、あくまでλつながりなだけ

第28章

ロマンティック・パージング

— @xhl_kogitsune

本章は、某 LLVM 狐本[*1](の表紙) に端を発した非公式妄想です。キャラ設定・名前等は公式設定[*2]とは異なりますのでご注意ください。他所の実在の人物 (勿論きつねさん含む)・言語処理系等とは関係ありません。

キャラクター及びストーリーは『簡約! λカ娘 (4)』所収の「インターフェース」の続きにあたりますが、技術的内容は前作とは独立となっています。

登場人物紹介:

キヒメ : 金髪バックエンドきつねさん
コヨネ : 黒髪フロントエンドきつねさん

二人は仲良し。

28.1 ことのはじまり

「……ねえコヨネ、えるあーる、って、なに?」

はじまりは、キヒメのそんな一言だった。その一言で頭が一瞬真っ白になって、それまで何を話していたか忘れてしまった。

「……え?」

完全に固まりつつも、なんとか声を絞り出すわたし。

「……え?」

自分が何か変なこと言ったのか、と、首をかしげるキヒメ。かわいい。

「……おしおき確定」

「え? わたし、何も悪いことしてないよ!?」

落ち着こう。そう思って私は湯呑みに手を伸ばす。晴れたおだやかな昼下がり。ともすると殺風景になりそうな合理性と、かわいらしさが同居している空間 —— キヒメの部屋で、私たちは優雅にお茶をしていた……はずだったのだ。

「ありえないわ! バックエンドとはいえ、コンパイラやってて LR 知らないとか!」

落ち着けなかった。

「えへへ……よく分からないけど、何の話?」

「LR(k) 構文解析(パージング)よ!」

「あっ……わたし、構文解析は授業も受けたけど、あまり得意じゃなくて……」

「得意じゃないってレベルじゃ……!」

「……赤点で……」

「でもっ、さすがに言葉くらいは……」

「……れいてん、でした……」

「……」

[*1] 柏木餅子, 風薬 (著), 矢上栄一 (表紙): **3 日で出来る LLVM**, MotiPizza (2012).
http://motipizza.com/catalog/c82
http://d.hatena.ne.jp/motipizza/20120724
http://d.hatena.ne.jp/sabottenda/20120728

[*2] http://motipizza.com/member

第29章

HaskEll Shaddai

— @fumieval

```
data Person = Person
    { name :: String, self :: Object, endurance :: Int, equipment :: Equipment }

data Object = Object
    { position :: Vector3
    , velocity :: Vector3
    }

data Equipment = Equipment
    { sword :: Object
    , body :: ArmorType
    }

data Vector3 = Vector3
    { x :: Float
    , y :: Float
    , z :: Float
    }
addV3 :: Vector3 -> Vector3 -> Vector3
```

「そんなレコードで大丈夫か？」
「大丈夫だ、問題ない」

```
enoch = Person
    { name = "Enoch"
    , endurance = 2
    , equipment = Equipment
        { sword = Object
            { position = Vector3 0 0 0
            , velocity = Vector3 0 0 0 }
        , body = Bronze}}
```

```
modify $ \e -> put e { endurance = endurance e - 1   }
```

「グゥッ！」

第30章

λカ娘探索 2?

— @ark_golgo

30.1 プロローグ

「……私の 1 目勝ちじゃなイカ？」
「そのようじゃな。ヨセで細かくなったとは思っておったが、逆転には至らなかったようじゃ。」
「3 子で初勝利でゲソ！　はぁ。ふらふらでゲソ！」
「この半年で置石 2 つ分くらい強くなったようじゃの。いやはやたまげたわい。お主と打ち始めた当初は 4 子でも楽勝だったがの。」
　λカ娘は　C84 のころからある老人と囲碁を打つようになった。当初は 4 子のハンデ (段級位差にして 4 段級差に相当) をつけてもらってもまるで勝てなかったが、最近は 3 子でもいい勝負が出来るようになっていた。そして、今回 3 子で初勝利を収めた。
「やっぱり囲碁ソフトと打ちまくったのが良かったんじゃなイカ？」
「囲碁ソフト？　はて、儂が昔試してみた囲碁ソフトはあまりに弱すぎて話にならなかったが……。最近はお主の相手になるくらい強いソフトが出ているのかね? 」
「そうでゲソ。ここ数年で囲碁ソフトは急激に進歩したでゲソ。 最近は私より強くて、トッププロに 4 子で勝ったりしているでゲソ！」
「ほう、そうなのか。面白そうじゃの。儂も一つ買ってみるか。」
　さて、最近強くなったという囲碁ソフトについて、ここでλカ娘に解説してもらうことにしよう。旧来の囲碁ソフトとの比較を交えながら、強くなった理由を語ってもらおう。

30.2 囲碁のルールおさらい

「囲碁のルールを説明するでゲソ。囲碁のルールは細かいものまで説明していると結構大変なので、今回の記事に関係ある部分だけ説明するでゲソ！　以下の 4 つを把握しておけば十分でゲソ。」

1. 黒白交互に盤の交点に打つ。1 手目は黒から (図 30.1)。[*1]
2. 相手の石の四方を囲むと取れる (図 30.2)。
3. 自分から石を取られるような手は打てない (自殺手の禁止) が、それ以外は空いている場所のほぼどこでも打てる。そのため、自由度が非常に大きい。ただし一見自殺手に見えても、先に周りの相手の石を取れる場合は打てる (図 30.3)(『コウ』という概念があって、厳密には打てない場所もあるが、今回は説明しない)。
4. 相手より大きな陣地を囲えば勝ち (図 30.4)。陣地とはそれぞれの石または盤端で四方を囲まれた石の無い領域のこと。図は数えやすいように 9 路盤 (練習用の小さな盤) で示している。黒と白の丸は実際の石で、四角はそれぞれの陣地を示す。四角の所には実際には何も置かれていない。この場合、黒の陣地は 36 目 (目(もく)とは地の単位)、白の陣地は 23 目で黒の 13 目勝ち。λカ娘が勝った対局は、 1 目だけλカ娘の陣地が大きかったということ。

[*1] ただし置き碁 (ハンデ戦) の場合は黒があらかじめ石を何個か置き、その後白から打ち始める。λカ娘が勝った対局は、黒のλカ娘が先に 3 個石を置いていた。

第31章

僕のカノジョはスナッチャー

— @master_q

31.1 川のほとり

『下におろして、垂直にまっすぐ持ち上げて敵を打ち抜くんでゲソ。ここ、両脇が大事なんでゲソ。そーして、打つべし、打つべし、打つべし、ぐるーんでゲソー!』
「なんなの、それ？」
『スナッチを制する者は、世界を制すでゲソ。最初のターゲットを決めるのが難しいんでゲソ』
「またサバゲーの話？」
『ん？ 何やってんでゲソ？』
「型推論の宿題」
『なんで大学でやんないんでゲソ？』
「かっこう悪いから」
『ワシのもやってくれなイカー』
「Dvorak 配列のままだよ」
『ぬ？ おぬしもスナッチやればいいじゃなイカ。なんでいつも ThinkPad 持ってるんでゲソ？』
「自分こそ、なんでいつもここに来るんだよ。」
『それはでゲソね、…あれ、なんででゲソ？』
「お気楽だね。イカ墨ください」
『出してないでゲソ』
「あのさ、なんでいつもこんなこと…」
『人間の指はおもしろいでゲソー』
「ん？」
『ワシはこうしてないと溢れちゃうじゃなイカ』
「溢れちゃうって、どうなっちゃうんだ？」
『……きっと…すごいことになるでゲソ……』

すごいことなんてない。ただ当たり前のことしか起こらない。僕達の大学から見えるオンボロなビル、ソフトハウス–メタセピのオフィスができたとき、プログラマたちは大騒ぎだった。毎日決まった時刻に鳴り出すスナッチのメロディーが、僕にはなんだか不吉なサイレンのように聞こえる。それはうっすらと広がり、世界を覆っていく。

31.2 カノジョのこと

僕のカノジョはスナッチャーだ。スナッチャーというのは他の生物の一部を置換して自分の体内に取り込むエイリアンの名前だ。いつから彼等がこの世界に現われたのかはわかっていない。とにかく現在では僕たち人間とスナッチャーが同居して暮していた。

僕が大学で授業を受けていると、カノジョが突然声をかけてきた。どうも授業の宿題が解けないらしい。僕は他人と会話するのが苦手なのでおっくうだったが、簡単に鍵となるアイデアを伝えた。次の日もその次の日もカノジョは僕に声をかけてきた。そのうち、僕達は毎週いっしょにハイキングにいくようになった。毎週どこにハイキングに行くのか、実のところそのネタを考えるのも僕にはおっくうだった。紙と鉛筆がそ