目次

第1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。

1.1 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。

[*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

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。

そんな単調な日々の終わりは突然やってきた。わたしがいつものように中間表現を淡々と流していると、突然背中に冷たい感触が走った。

「……ひゃあん!?」

触られた! いきなり背中を触られた!

驚いて —— というより激しい怒りで —— 振り向いたわたしは、そこに一人のフロントエンドが立っているのを見た。わたしが処理していた中間表現を持ってきた奴だ —— 確か。

「ごめんなさい、そんなに驚くとは思わなくて……でも、内部インターフェースがあったから、なんなのかなー、って気になって」

あんまり悪いとも思っていなさそうな、のほほんとした態度の黒髪の女。それに余計にカチンと来たわたしは、

「だからって何で触るのよ! そもそも内部インターフェースなんだからアンタみたいなフロントエンドのためのものじゃないの! 他人の undocumented なトコロに触るなんてサイテー!」

そう叫ぶように言い捨てて、わたしは処理途中だった中間表現をそいつに突き返して、その日はそのまま帰ってしまった。この女が —— 名前は後から知ることになる —— コヨネだった。

後で聞いたところによると、この時のコヨネにとってのわたしの印象は「かわいい」、だったらしい。

『驚きと怒りのあまり顔を真っ赤にして涙目になりながら非難してくる、金髪のバックエンド。かわいい』

……ひどい女だ。

この日から、この女はわたしにちょっかいを出してくるようになる。

翌日もまた、彼女は中間表現を持ってやってきた。中間表現なんて、別にネットワーク送信で済むのだし、昨日の今日で手で渡しにくるという時点で警戒せざるを得ない。背中を向けないように注意しつつ、無言で受け取り、開く。

……何考えてるの、この女。

そこには、ぱっと見 nop しか書いていなかった。そもそも、中間表現において nop とか普通は使わない。なのに、nop しかないこれは何だ。嫌がらせか? いつもどおり淡々と処理しようとしたわたしはやや面食らい、一瞬手を止めてしまう。しかし、それは何か癪な気がしたので、そのまま nop の列をスルーし、さっさとほぼ空のコンパイル結果をつき返して後は無視することに決めた。

しかし、翌日以降も奴は妙な中間表現を —— それも毎日 —— 持ってきた。嫌がらせに違いない。気にしたら負け。わたしはそう自分に言い聞かせ、完璧に通り一遍のコード生成だけをして、さっさと突き返す。いつしか、そんなよく分からない日課のようなパターンが出来てしまっていた。

ある日、彼女はみかん箱を持ってきた。……今度は何だ。新手の嫌がらせか? しかも、みかん箱をわたしの机の上に乗せて、じっと見られても困るんだけど。

「……何これ」

仕方ないので、またろくでもない嫌がらせか早くカエレ! って気持ちを込めて聞いてみる。すると、中間表現のコンパイルを頼みたいのだと言う。いつものと違ってこれは普通のユーザープログラムを普通にコンパイルしたものだ、とも。

……みかん箱で中間表現持ってくる奴をわたしは初めて見た。手渡しで中間表現を持ってくるフロントエンドはこいつ以外にもたくさんいるが、それは言ってみればスクリプト言語におけるワンライナー

第2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
```

2.1 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
```

[*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

```
      | otherwise = return 0
```

```
/* File: CMain.c */
#include <stdio.h>
#include "HsFFI.h"
#ifdef __GLASGOW_HASKELL__
#include "Fib_stub.h"
#endif

int main(int argc, char *argv[])
{
    int i;
    hs_init(&argc, &argv);
    for (i = 0; i < 30; i++) {
        printf("%d\n", fib(i));
    }
    hs_exit();
    return 0;
}
```

C 言語の main 関数から Haskell で書かれた fib 関数を呼び出すようにしてみたでゲソ。前号の記事[*4] で判明したように、Haskell から標準出力すると実装が大きくなってしまうので、今回は C 言語の printf で印字することにしたでゲソ。

2.2 ダイエット指標

単に小さい ELF バイナリを作るのではく、今回は「GHC コンパイラが出力した ELF バイナリをダイエット」するのでもう少し詳細なルールを決める必要があると思うでゲソ。一つの軸で評価するのではく、次の 3 つの指標をバランス良くダイエットしようと思うでゲッソ!

2.2.1 指標 1: text/data/bss セクションの合計サイズをダイエット

ELF を strip したりするのは GHC とあまり関係ないので、text/data/bss セクションを合計したサイズを小さくすることを目標にするでゲソ。減量前のプログラムをコンパイルしてサイズを調べてみると

```
$ make
gcc -I/usr/lib/ghc/include -c CMain.c
/usr/local/ghc7.6.1/bin/ghc -O2 -c Fib.hs
/usr/local/ghc7.6.1/bin/ghc -O2 -no-hs-main CMain.o Fib.o -o FibHs
$ size FibHs
   text    data     bss     dec     hex filename
2784310  290592   47960 3122862  2fa6ae FibHs
```

[*4] 簡約!?ﾚｶ娘 (算) http://www.paraiso-lang.org/ikmsm/books/c82.html

第3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
```

第4章

OCaml で printf じゃなイカ? — @xhl_kogitsune

4.1 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

[*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5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6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――」

6 月のジメジメした季節。結局あの時渡されたニコリのパズル雑誌は、経験の浅い私にはそんなにすぐに解けるはずもなく。それどころか、未だに半分以上残っていました。先輩たちが毎日お茶をしているのを横目に、私の気分まで、梅雨空のように曇るのでした。

「あーっ。めんどくさい！」

そう、めんどくさいのです。SAT ソルバは、その名の通り、命題論理式の充足可能問題しか解けません。変数は真か偽かの 1 ビットの情報しか載せられず、整数を扱うのにも、ビットを束ねて、加算器をつなげて、論理回路を一から書いているような状況です。さらに、その際にかなり工夫して書かないと、サイズが膨れ上がって、思うように大きいサイズの問題が解けなくなってしまいます。

「苦戦しているようだな」

見かねて澪先輩がやって来ました。

「はい、思うようにコードが書けなくて———」

「うーむ、なるほど———」

私のコードを覗いて、なにやら独り言を呟いて、

「———そろそろ、頃合いかな。梓、」

「は、はい！」

「お前に、SMT を教えよう」

え、えすえむ、ティー？また変な名前の———。

「あずにゃん、そういうのじゃないよ」

「うわぁ、唯先輩。そういうの考えてないです！」

相変わらず、唯先輩には心を読まれているようです。

「スモール・ミディアム・トール、でもないわよ」

「ムギは飲み物の話ばかりだな」

SAT は、もう、たくさん。

「Satisfied Modulo Theories。簡単に言えば、SAT に別の理論を追加したものだ」

何かを追加する…？

「SAT は Boolean ロジックしか扱えない。だから、せっかく高性能になった SAT ソルバを利用しようと思っても、それを Boolean のコードにエンコードしなければいけない」

そう、それがめんどくさいんです。 SAT、めんどくさくて、たいへん。

「ただ、めんどうなだけならそこまでの問題ではないぞ。例えば、32 ビットの整数の足し算をするには、全加算器を 32 個つなげることになる。全加算器は、XOR を含む論理回路として表現できるが、XOR はそのままでは表せないから、これも CNF に変換すると、かなりサイズが大きくなってしまうんだ」

「足し算 1 つで、数百節になってしまうと、せっかく数十万規模の CNF を扱えても、実際に扱える整数問題のサイズは、かなり目減りしてしまうわね」

そう、それで私も困っていたところでした。

「まあ、中には CNF に加えて XOR 節を扱えるようにした、CryptoMiniSat みたいな処理系も存在するが、それは今回は置いておこう」

「CryptoMiniSat…暗号解析用のソルバなんですか？」

「いや、これは汎用の SAT ソルバで、特に暗号向けというわけではないぞ。暗号ルーチンに XOR が多く含まれるというのは無関係ではないだろうけど」

全加算器にも XOR は多く使われるから、整数の計算も高速化されそうだと思いました。

「多くの最新の SAT ソルバは、XOR のようなパターンは自動で認識して、特別な高速化が施されているから、今現在ではあまり考えることはないかな」

「へぇ、SAT ソルバってすごいんですね」