Joy of Modularisation
部品化の楽しみ

Kazu Yamamoto

IIJ-II

kazu@iij.ad.jp

The most precious resource is
programmer's time.
So, let's learn a new language
to save our time

最も貴重な資源は
プログラマーの時間です。
新しい言語を学んで
時間を節約しよう。

"Beating the Averages"

Paul Graham

http://www.paulgraham.com/avg.html

「普通のやつらの上を行け」

ポール・グレアム

http://practical-scheme.net/trans/beating-the-averages-j.html

The Blub Paradox

At least as a kind of social convention,
high-level languages are often
all treated as equivalent. They're not.

Languages fall along a continuum of abstractness,
from the most powerful all the way down
to machine languages, which themselves vary in power.

Blub言語のパラドックス

少なくとも社会的な慣習として、高級言語は対等
であるとされることが多い。それは間違いだ。

プログラミング言語はそれぞれが、
機械語から最も力のある言語までの
連続した抽象度のスペクトルのどこかに位置するのだ。

As long as our hypothetical Blub programmer
is looking down the power continuum,
he knows he's looking down.
Languages less powerful than Blub are obviously
less powerful, because they're missing some feature
he's used to.

このプログラマ氏がパワーのスペクトルを
見下ろしている時、彼にはそうしているという自覚がある。
Blub よりも力の弱い言語は、明らかに力が弱い。
彼が慣れ親しんだ機能が無いからだ。

But when our hypothetical Blub programmer
looks in the other direction, up the power continuum,
he doesn't realize he's looking up.
What he sees are merely weird languages.

What he sees are merely weird languages.
He probably considers them about equivalent in power to Blub,
but with all this other hairy stuff thrown in as well.
Blub is good enough for him, because he thinks in Blub.

しかし、このプログラマ氏が反対の方向に目を転じた時、
彼は自分が見上げているのだということに気付かないのだ。

彼が目にするのは、変てこりんな言語ばかり。
多分、それらはBlubと同じくらい強力なんだろうけど、
どういうわけかふわふわしたおまけがいろいろついているんだ、
と思うだろう。彼にとっては Blub で十分なのだ。
何故なら彼は Blub で考えているから。

Hairy stuff?

ふわふわしたおまけ？

Hairy stuff that you don’t know

Lazy Evaluation

あなたの知らないふわふわしたおまけ

遅延評価

Perl

Strict evalutation -- 正格評価

```
% perl -e 'func(1 / 0)'
→ Illegal division by zero.
```

Haskell

Lazy evaluation -- 遅延評価

```
% ghc -e 'head [2, 1 `div` 0]'
→ 2
```

So, what?

だから何？

"Why Functional Programming Matters"

John Hughes

http://www.md.chalmers.se/~rjmh/Papers/whyfp.html

Lazy evaluation allows termination conditions to be separated from loop bodies - a powerful modularisation.

「なぜ関数プログラミングは重要か」

ジョン・フューズ

http://www.sampou.org/haskell/article/whyfp.html

遅延評価はループの本体から終了条件を切り離す。これは強力な部品化の手法である。

Modularisation is important

Chair

Carve it out of a solid block of wood

vs

Seat + back + back + glue

部品化の重要性

椅子

丸太から切り出す

vs

底＋背＋足＋ノリ

Lazy evaluation → modularisation

遅延評価 → 部品化

Replicate

- Spec

```
replicate 5 '*' → "*****"
```

- Loop

```
replicate (n, x) {
   var ret = "";
   for (var i = 0; i < n; i++)
       ret = push(x,ret);
   return ret;
}
```

- Lazy evaluation

```
repeat '*'
→ ['*','*','*','*','*', …]

take 5 ['*','*','*','*','*', …]
→ ['*','*','*','*','*']
→ "*****"
replicate n c = take n (repeat c)
```

Netmask

```
iterate f n
→ [n, f(n), f(f(n)), f(f(f(n))), ...

shiftL n = shift n 1

iterate shiftL 0xffffffff
→
[255.255.255.255,
 255.255.255.254,
 255.255.255.252,
 ...
 128.0.0.0,
 0.0.0.0,
 0.0.0.0,
 0.0.0.0, ...

take 33 (iterate shiftL 0xffffffff)
```

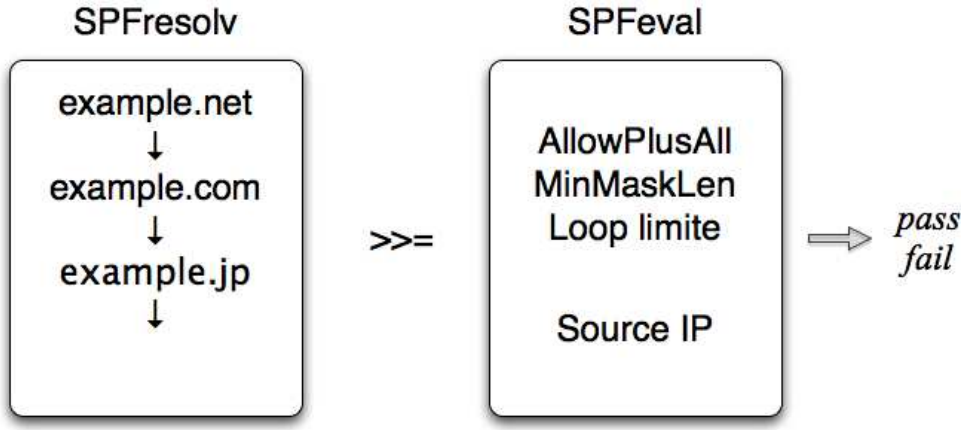
Handling SPF securely

- SPF RR of Example.com
  v=spf1 +ip4:192.0.2.1 +ip4:192.0.2.2 redirect=example.jp
- SPF RR of Example.jp
  v=spf1 +ip4:192.0.2.3 -all

- Rejecting +all or +0.0.0.0/0 in SPF
- Avoiding infinite loop of "redirect" / "include"

SPF with Lazy evaluation

- SPF resolver
  - Resolving and parsing SPF RR
  - Infinite trace of "redirect" and "include"
- SPF evaluator
  - Evaluating SPF RR until a result is obtained
  - Rejecting +all and short mask length
  - Avoiding infinite loop with a limit counter
- Gluing with lazy evaluation



Other hairy stuff

High-order function
Currying
Reasoning about programs
List comprehensions
Type inference & check
Monad

他のふわふわしたおまけ

高階関数
関数のカリー化
プログラムの論証
リスト内包表記
型推論と型検査
モナド

Let’s learn a new language!

さあ、新しい言語を学ぼう！