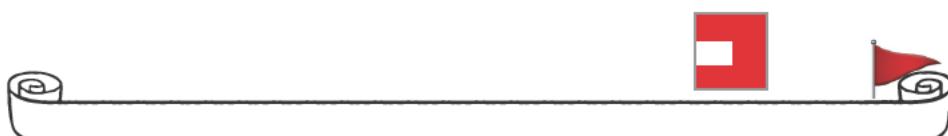


# WORD

FROM COLLEGE OF INFORMATION SCIENCE



"農水省も「一太郎」より WORD! 号"



グミ  
jlreq.cls を信じろ  
Neovim の車窓から～Lua のある日常～  
量子コンピュータなんかに絶対屈したりしない  
Road to Princess ☆ tree  
シャドウベース

2018 | 44

# 目次

グミ .....	わだぐみこ	003
jlreq.cls を信じろ .....	ひだるま	008
NeoVim の車窓から～Lua のある日常～ .....	びしょ～じょ	012
量子コンピュータなんかに絶対屈したりしない .....	みみずのひもの	021
Road to Princess ☆ Tree .....	みみずのひもの	038
シャドウバス .....	ドバ	049

# グミ

文 編集部 わだぐみこ

## 1 はじめに

### 1.1 ことのはじまり

私はグミが大好きです。そこで、この世に存在する様々なグミを紹介し、グミの良さを広めようと決意しました。向かったのはイオンつくば駅前店。売り場にあった全てのグミを1種類ずつ購入し、私はいそいそと帰宅したのでした……。

### 1.2 表の見方

評価は5点満点で、私とある協力者がつけた点数の平均をとっています。評価項目は味総合・酸っぱさ・甘さ・食感の4項目です。さらに、表示されている重量と実際の重量が同じかどうか調査し、実際の重量に基づいて100gあたりの値段を算出しました。

### 1.3 お詫び

実際の重量を計測したばかりは1g単位でしか重さを量れません。それにもかかわらず、実際の重さに小数点以下がついている項目があります。何故でしょう。量る前につまみ食いをしてしまい、1粒の重さが同じだと仮定して残った分から計算したのです。ごめんなさい。

## 2 果汁100%編

果汁100%を謳うグミたちです。原材料の100%が果汁という訳ではないのであった。

### 2.1 味覚糖 コロロ マスカット味

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	1.5	4.0	4.0	48	48.45	295

グミ全体が薄い皮で包まれていて<sup>\*1</sup>、本物のブドウを食べた時のようなプチプチ感が再現されています。皮が口の中に残ることと100gあたり295円と牛肉並のお値段であることだけが残念。パッケージを見ると20%増量とあったので以前はもっと高価だったことが窺えます。なお、さらに高級になったcororo<sup>\*2</sup>というシリーズもあるようです。

<sup>\*1</sup> 噛まずに口の中で転がし続けると皮部分とグミ部分が分離します。やってみよう。

<sup>\*2</sup> 阪急百貨店うめだ本店で販売されているようです。

## 2.2 明治 果汁グミ おいしくコラーゲン

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	1.5	4.0	4.0	81	81	237

コラーゲンが6000mg入っているという健康志向らしきグミです。ザクロ味のことですが私は本物のザクロを食べたことがありません。実家にザクロの木があったという人に感想を聞くと、ジューシーでザクロ感<sup>\*3</sup>が再現されているとのことです。美味しいです。

## 2.3 明治 果汁グミ 温州みかん

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
5.0	1.5	4.0	4.5	51	51	186

冬にこたつで食べる甘いミカンの味がします。表面の柔らかさもミカンにそっくりで、いつでも冬気分が味わえますね。噛み碎くと口の中でミカンを搾ったジュースができあがります。

## 3 酸っぱい編

周りに酸っぱい粉がまぶされているのが特徴。

### 3.1 ブルボン フェットチーネグミ イタリアンピーチ味

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	2.5	4.5	4.0	50	52	183

外側の粉の酸っぱさが丁度良く、しかも口の中でその酸味が長続きします。グミ自体も甘酸っぱく、桃の表面と中心の間の部分の味に似ています。気温が35度以上になると溶けることがあるそうなので注意しましょう。部屋に放置していたら少しく述べていました。

### 3.2 カンロ ピュレグミ グレープ味

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.0	4.5	4.5	4.5	56	58	116

外側のパウダーの酸っぱさがグミ本体の甘さを引き立てています。グミ自体は他と比べて甘めで、巨峰の酸っぱさを減らしたような味です。外側がどうしても食べられないという人は洗ってから食べましょう。洗いすぎに注意。

<sup>\*3</sup>「ブチっ、ああ、これがザクロか……そんなに甘くないな。」という感覚。

## 4 噛みごたえ編

噛みごたえ・弾力をウリにしているグミたちです。1粒で長時間味わえます。

### 4.1 ノーベル ちび SOURS

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	2.0	4.0	4.0	80	81	210

グレープ味・温州みかん味・白桃味・グリーンアップル味の4種類のグミが入っています。表面の粉のおかげでグミ同士がくっつかないようになっていて、手もベタベタしません。味は甘めで、名前に SOUR と入っている割にはサワー感が少ないです。

### 4.2 明治 GOCHI グレープ味

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
3.5	2.0	3.0	3.5	52	54	176

周りに粉がまぶされていて酸っぱい系かと思いきや、粉の正体はザラメです。外側と内側で違う味を楽しむといけど酸っぱいのは苦手、という人におすすめ。噛むとザラメが砕けてシャリシャリします。

### 4.3 カバヤ TOUGH GUMMY

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
3.5	4.5	3.0	3.5	100	102	167

コーラ味・ソーダ味・ジンジャーエール味の3種類のグミが入っています。商品名に「TOUGH」と入っているだけにかなり固めの食感で、1回では噛みきれません。外側の粉の酸っぱさが炭酸感を醸しだし、さらにグミ本体もそれぞれの飲み物の味をよく再現しています。特にジンジャーエール味は後味のショウガ感がカナダドライ（味がリニューアルされた後の方）にそっくりです。

### 4.4 カンロ カンデミーナ

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.0	4.0	3.0	3.5	72	73	170

コーラ味・サイダー味・スポーツドリンク味の3種類のグミが入っています。表面には酸っぱい粉がふんだんにまぶされていて、その量は袋の底に落ちた粉が溜まるほどです。食感は非常に固く、かみ続けると唾液と相まってジュースらしくなってきます。スポーツドリンク味はグリーンダカラ、サイダー味は三ツ矢サイダー、コーラ味はコカコーラとドクターペッパーの中間の味がします。

## 4.5 三菱食品 HARIBO Happy Cola

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	1.0	3.0	4.5	100	101	222

噛みごこちといいコーラ味の美味しさといい、ハリボーはグミの王者にふさわしい。ハリボーはコーラ以外にも種類があり、よく知られているものはフルーツ、その他にもサクランボやベリー系もあり、どれも美味しいです。色々な種類が売られているのはヴィレッジヴァンガードですが、コストコではフルーツ味が透明のパケツに980g入りで売られています。

## 4.6 明治 コーラアップ

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
*4.5	1.0	3.0	4.5	100	101	222

グミを買いに行ってハリボーが無いときについつも買っています。つまりジェネリックハリボー。食感は硬くて噛み応えがありますが、その分ややゼラチンっぽさが強く、味が薄めに感じます。ジェネリックなだけあってハリボーよりコスパが良いです。

## 5 その他編

### 5.1 トップバリュー プチグミフルーツアソート

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
1.5	5.0	1.0	3.0	80	82	180

レモン味・グレープ味・ピーチ味・メロン味の4種類のグミが入っているはずでした。袋を開けると噛みかけのガム（グレープ味）のようなにおいが漂ってきます。おそるおそるメロン味を口にいれてみると、なんとキウイの味が。続いてグレープ味もキウイ味。ピーチ味は桃の中心部の酸っぱいところの味がしました。レモン味は普通に酸っぱいレモンの味でした。キウイが大好きな人におすすめ。

### 5.2 トップバリュー フルーツグミ リンゴ

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	1.5	4.0	4.0	52	54	156

一つ前の項で紹介したものと同じトップバリューフルーツグミ。これもキウイ味かと思いきや、美味しいリンゴジュースの味がします。リンゴ特有の酸っぱさと甘さのバランスがよく再現されていて美味しいです。100gあたり156円と今回紹介した中で一番のコスパ。箱買いをおすすめします。

### 5.3 明治 Poifull

味総合	酸っぱさ	甘さ	食感	表示重量(g)	実際の重量(g)	100gあたりの値段(円)
4.5	1.5	5.0	4.0	80	83	218

グレープ味・ラズベリー味・レモン味・キウイ味の4種類のグミが入っています。キウイ味とラズベリー味は珍しいですね。弾力があり、とてもジューシーです。味は本物の果物の味に近いですが、酸っぱさは控えめで食べやすくなっています。ちなみにコラーゲンが6200mg<sup>\*4</sup>入っているそうです。

## 6 おわりに

家に帰ってきてすぐは、「ユーチューバーになった気分だ<sup>\*5</sup>」と大喜びでグミの写真をTwitterに載せ、一心不乱に貪り食っていましたが、4種類目を超えたあたりからだんだん辛くなってきました。顎にニキビができて痛いです。と言いつつも、今私はグミを食べながらこれを書いています。グミは美味しいなあ。一番のおすすめはコロロとハリボーです。



図1: 買ってきたばかりのグミたち。GOCHIの封が既に開いているのは何故だろう。

\*4 なんと、「果汁グミ おいしくコラーゲン」よりも多い！

\*5 湯船をグミで埋めるとかはやってません。

# jlreq.cls を信じろ

文 編集部 ひだるま

## 1 はじめに

皆さん、 $\text{\LaTeX}$  を使うときクラスファイルは何を指定していますか？多くの人は `jsclasses` を利用していることでしょう。コミュニティ版に開発が移ったことで `tbook` なども久々に修正が入り<sup>\*1</sup>、既存クラスファイルの使い易さが上がっているようです。

そんな中、新しいクラスファイルも登場しています。`p\LaTeX`、`Lua\LaTeX`、`Xe\LaTeX` に対応した `bjaxclasses`<sup>\*2</sup> や、今回紹介する `jlreq` です。

### jlreq とは

W3C では多くの事柄が策定されていますが、その中に日本語組版についてのものがあります<sup>\*3</sup>。以下 `jlreq` の `texdoc` から引用です。

### これは何？

日本語組版処理の要件の実装を試みる `LuaTeX-jja` / `pLaTeX` / `upLaTeX` 用のクラスファイルと、それに必要な JFM の組み合わせです。

### 1.1 提供されるもの

クラスファイル `jlreq.cls` と、横書き `LuaTeX-jja` 用の JFM である `jfm-jlreq.lua` が用意されています。また、縦書きの JFM や `pLaTeX` / `upLaTeX` 用の JFM を生成するいくつかのスクリプトがあります。

以上です。JFM は和文フォントメトリック<sup>\*4</sup>のことです。

つまり、 $\text{\LaTeX}$  で W3C の仕様に沿いつつ良い感じに日本語が書けるクラスファイルが登場した、ということです。多分。

## 2 インストール

CTAN 収録パッケージなので、`TeXLive2017` の日本語周りのパッケージが入っていれば何もしなくても入っていますが、見つからない場合 `tlmgr install jlreq` で入ると思います。

---

<sup>\*1</sup> <http://acetaminophen.hatenablog.com/entry/new-plateX-20170505-01>

<sup>\*2</sup> これを基に WORD の新クラスファイルを作る試みがあるとかないとか。

<sup>\*3</sup> <https://www.w3.org/TR/jlreq/>

<sup>\*4</sup> 文字の高さ、幅、スペースなどの値を定義する情報。

### 3 説明

基本的には公式の説明を読んでいくのが確実です。`texdoc jlreq`で読むことができます。他には `jlreq.cls` の作者 abenori さんのブログ<sup>s</sup>が参考になるでしょう。

幾つか代表的な機能を紹介します。

#### 3.1 オプション

```
1 \documentclass[<option>]{jlreq}
```

で指定するオプションについてです。かなり多岐に渡るので主要なもののみ説明します。

- `yoko, tate` 横書き、縦書きを設定します。デフォルトは `yoko` です。
- `paper=[紙サイズ名]/[<横寸法>,<縦寸法>]` 紙サイズ名は `a0~a10, b0~b10, c2~c8` を指定できます。もしくは直接寸法を指定できます。
- `fontsize=<寸法;Q>` 欧文フォントサイズです。デフォルトは `10pt` です。
- `jafontsize=<寸法;Q>` 和文フォントサイズ。デフォルトは欧文に合わせて `10pt` です。
- `jafontscale=<実数値>` 欧文フォントと和文フォントの比です。上 2 つが両方指定されているときは無視されます。デフォルトは `1` です。
- `line_length=<寸法;zw,zh>` 1 行の長さです。
- `number_of_lines=<自然数値>` 1 ページの行数です。

クラスオプションで指定するものの他、`hyperref` パッケージのように `\jlreqsetup` で指定するものもあります。

#### 3.2 section

見出しに副題がつけられます。書き方は

```
1 \section{見出し}[副題]
```

となります。また、書式設定用の命令も用意されています。

#### 3.3 sidenote

傍注（縦組みの場合は脚注）を出力します。

#### 3.4 warichu

割注を出力します。割注は後述のサンプルにあるように 1 行に 2 行を縮め書くような書式です。

---

<sup>s</sup> <http://abenori.blogspot.jp/search?q=jlreq&by-date=true>

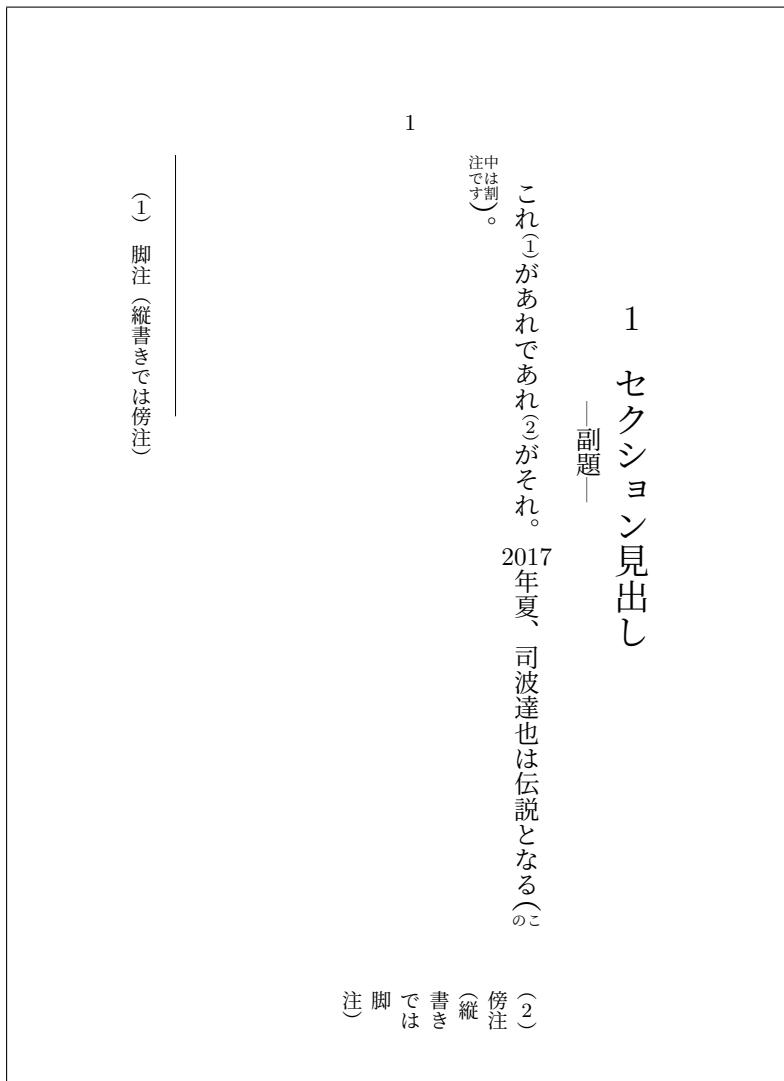
### 3.5 tatechuyoko

縦組みの中で横組みをしたいときに用います。tbook などにおける\rensushiのような使い方ができます。

## 4 サンプル

以下、サンプルになります。

```
1 \documentclass[tate, paper=a6, notitlepage, article]{jlreq}
2 \usepackage[sourcehan]{luatexja-preset}
3
4 \begin{document}
5 \section{セクション見出し}[副題]
6 これ\footnote{脚注（縦書きでは傍注）}があれであれ\sidenote{傍注（縦書きで
    は脚注）}がそれ。\tatechuyoko{2017}年夏、司波達也は伝説となる
7 \warichu{この中は割注です}。
8 \end{document}
```



pdf にするとこうなります。

## 5 おわりに

端折りましたが jlreq クラスの紹介でした。かなり自由かつ簡単に望みの日本語組版ができるクラスなのではないかなと思うので僕は推していきます。小説とか組みたいよねやっぱり。SourceHan<sup>\*6</sup>も同時推ししているので、興味が出たら是非調べてみてください。

<sup>\*6</sup> Adobe の出したフォント。かなり緩いライセンスで使えます。

# Neovim の車窓から～Luaのある日常～

文 編集部 びしょ～じょ

## 1 はじめに

こんにちは、びしょ～じょです。(前略)(中略)(後略) Neovim してますか? Lua してますか? Neovim の Lua インターフェースが 2017 年 5 月 8 日に master に merge されました \*1。

では、やっていきましょう。

## 2 インストール

あなたがタイムトラベラーでなければ、Neovim のリポジトリ \*2 から master を clone してビルドすれば問題ないと思います。ちなみに今記事を書いている ArchLinux 上では MessagePack のビルドでコケてしましました \*3。悲しいですね。

詳しくは wiki の *Install from source*\*4 などを参考にしてください。あるいはパッケージマネージャーなどでインストールできれば、v0.2.1 以降なら Lua インターフェースが備わってるはずです。この記事がデブロイされる頃には pacmanあたりでインストールできる気がします \*5。

## 3 使い方

詳細は :help lua で見られます。

### good to know

Lua 環境は各 Neovim プロセス中で共有されます。なので一つの Neovim プロセス内に複数のタブやウィンドウがあっても、同一のグローバル変数を共有できます。

### Neovim API

vim.api というモジュールで Neovim の API にアクセスできます。

---

\*1 <https://github.com/neovim/neovim/commit/09f849b60000c2d401d82f2b2fb2badde5583658>

\*2 <https://github.com/neovim/neovim>

\*3 <https://github.com/neovim/neovim/issues/6846>

\*4 <https://github.com/neovim/neovim/wiki/Installing-Neovim#install-from-source>

\*5 2017/11/08 に Neovim0.2.1 が正式リリースされました、めでたい!!! <https://github.com/neovim/neovim/releases/tag/v0.2.1>

```
:lua print(vim.api.nvim_command([[echo "Hello, world"]]))
```

詳細は:`:help api`をご覧ください。

### 3.1 commands

```
:[range]lua {chunk}
```

引数の Lua チャンクを実行します。

```
:lua print("ok")
```

複数行書きたい時は here document を使います。

```
:lua <<LUA
local t = {a = 1, b = 2, c = 3}

for k, v in pairs(t) do
    print(k, v)
end
LUA
```

```
:[range]luado {body}
```

引数の`{body}`を `function(line, linenr) {body} end`という関数にして、各行の内容と行数を作った関数に適用します。この関数の戻り値(を `tostring`した文字列)が適用した行の文字列に置き換わります。戻り値が`nil`のときは置き換わりません。

ンー？ 使ってみましょう。

Vim の現在のバッファーが次のようになっています。

neovim current buffer

```
11 .....  
12 \subtitle{エンパワメント}  
13 \author{びしょ～じょ}  
14  
15 \begin{document}  
16 \chapter{Neovimの車窓から～Luaのある日常～}  
17 .....
```

ここでおもむろに

neovim command line

```
:12,16luado return ("% -3d %s"):format(linenr, line)
```

とすると

replaced current buffer

```
11 .....  
12 12 \subtitle{エンパワメント}  
13 13 \author{びしょ～じょ}  
14  
15 15 \begin{document}  
16 16 \chapter{Neovimの車窓から～Luaのある日常～}  
17 .....
```

おお、先頭に行数がつきました。おめでとうございます。range に 12～16 様目を指定してその行だけに適用しています。单一テキストファイルでソースコードとその解説を提出するタイプの講義で活躍しそうですね<sup>\*6</sup>。range で何も指定しなければバッファーの全行に適用されます。

```
:[range]luofile {file}
```

引数で指定された lua ファイルを実行します。`:lua`、`:luado` 同様に range を用いて適用範囲を指定することができます。

### 3.2 functions

`luaeval`

Lua のプログラムを表す文字列を受け取り、それを評価します。ちょうど Lua の `vim.api.nvim_eval` と対応になる感じですね。

---

<sup>\*6</sup> T 大学の情報科学類にはそういう形式の講義がいくつかある。

```
:echo luaeval('print("hello")')
```

## 4 実演

かんたんなファイルを作つてみます。ディレクトリーのコンテンツ一覧を表示したり一覧からファイルを開くことができます。

filer.lua

```

1 plugin = plugin or {}
2
3 if plugin.filer then
4   return
5 end
6
7 local api = vim.api
8 -- ファイルシステムを扱うモジュール lfs
9 local lfs = require('lfs')
10 local verror = api.nvim_err_write
11 local bufs = {}
12 local filer = {
13   bufs = bufs
14 }
15
16 plugin.filer = filer
17
18 -- [[
19 =====
20 /path/to/foo/bar
21 =====
22 のようなヘッダーを作成する
23 --]]
24 local function generate_header(path)
25   local eqs = ('='):rep(#path)
26   return {eqs, path, eqs, ''}
27 end
28
29 local header_height = 5
30
```

```
31 function filer.render_dirs(path) -- レンダリング関数 {{
32   path = path or lfs.currentdir()
33   local current_buf = api.nvim_get_current_buf()
34   local replace_tbl = {}
35   local fileAttrs = {}

36
37   -- normalize path name {{
38     -- パスの最後にスラッシュつけるマン
39     if not path:match('/$') then
40       path = path .. '/'
41     end
42
43     -- パスに含まれる `./` みたいなやつの処理
44     while path:match('%.%./$') do
45       path = path:match('^(..)/[^/]+%.%./$')
46     end
47
48     while path:match('%.//$') do
49       path = path:match('^(..)%./$')
50     end
51   -- }}}

53   -- ディレクトリーにあるアイテムのイテレーション
54   for file in lfs.dir(path) do
55     local name = file
56     local attr = lfs.attributes(path .. file)

58     if attr.mode == 'directory' then
59       name = name .. '/'
60     end

62     -- replace_tblにファイル名、fileAttrsにそのファイルのアトリビュートを保持する
63     table.insert(replace_tbl, name)
64     table.insert(fileAttrs, attr)
65   end

67   local header = generate_header(path)
68   -- ヘッダーをレンダリング
69   api.nvim_buf_set_lines(current_buf, 0, header_height - 1, nil, header)
70   -- ファイルを羅列
```

```
71 api.nvim_buf_set_lines(current_buf, header_height - 1, -1, nil, replace_tbl)
72 -- パッファーを読み込み専用にする
73 api.nvim_buf_set_option(current_buf, 'modifiable', false)
74
75 -- パッファーの情報を更新 {{{
76 bufs[current_buf] = {root = path}
77
78 for i = 1, #replace_tbl do
79   bufs[current_buf][i] = {
80     name = replace_tbl[i],
81     attr = fileAttrs[i]
82   }
83 end
84 -- }}}
85 end -- }}}
86
87 -- open file with vertical or horizontal
88 local openingMode =
89 setmetatable({
90   v = 'vnew',
91   h = 'new'
92 }, {
93   __index = function(_, k)
94     if not k then
95       -- default horizontal
96       return 'new'
97     end
98
99     return verror(("invalid mode '%s'):format(k))
100   end})
101
102 function filer.open(mode) -- カーソル下のファイル/ディレクトリーを開く
103   -- カーソルのある行を取得
104   local currentLine = api.nvim_win_get_cursor(api.nvim_get_current_win())[1] -
105     header_height + 1
106
107   if currentLine < 0 then
108     return verror('no file specified to open')
109   end
```

```

110 local current_buf = api.nvim_get_current_buf()
111 -- カーソル下のファイル情報を取得
112 local buf_consts = bufs[current_buf]
113 local cont = buf_consts[current_line]
114
115 -- ディレクトリーならレンダリングする
116 if cont.attr.mode == 'directory' then
117     api.nvim_buf_set_option(current_buf, 'modifiable', true)
118     filer.render_dirs(buf_consts.root .. cont.name)
119     api.nvim_buf_set_option(current_buf, 'modifiable', false)
120 else
121     -- ファイルなら新しいウィンドウで開く
122     api.nvim_command(opening_mode[mode] .. buf_consts.root .. cont.name)
123
124 local list_win = api.nvim_list_wins()
125 table.sort(list_win)
126 -- 操作を新しく開いたウィンドウに移す
127 api.nvim_set_current_win(list_win[#list_win])
128 end
129 end

```

ふむ。ファイルシステムを扱える Lua のメジャーなモジュールである `LuaFileSystem*7` と Neovim API を使用しています。名前衝突を避けたいので `plugin` という `table` を作ってそこに `filer` というプラグインを追加しました。やっとることの詳細はコメントに書いてあるので読んでみてください。

`filer.render_dirs(path)` でバッファーに引数のパスの、または引数なしで `pwd` のファイルコンテンツ一覧を表示します。`bufs` という `table` に Neovim バッファーに出力したファイルの名前とその属性を保存します。この `bufs` はインスタンスごとに共有し、`filer.open()` でこの `table` からファイル名を取り出し、ファイルを開くのか `filer.render_dirs()` を実行するのかを計算します。

ではさっそく使ってみます。

neovim command line

```
:luafile filer.lua
:lua plugin.filer.render_dirs()
```

なんということでしょう。現在のバッファーが `ls` コマンドの実行結果みたいになりました。

開きたいファイル/ディレクトリーの上にカーソルを移動し、

---

<sup>\*7</sup> <http://keplerproject.github.io/luafilesystem/>

```

1 ======<
2 /home/nymphium/works/github/wordarticles/WORD44/articles/nvim-lua/<
3 <
4 ./<
5 ../<
6 .gitignore<
7 CMakeLists.txt<
8 Makefile<
9 Makefile.in<
10 cmake/<
11 configure.ac<
12 latexmkrc<
13 main-lua.tex<
14 main-ptex.tex<
15 src/<
16 word-lua.cls<
17 main-lua.aux<
18 main-lua.fdb_latexmk<
19 main-lua.log<
20 main-lua.pdf<
21 img/<
22 main.tex<
23 main-lua.flst<
~
~
~
~
~
~
~
~
~
~
~
~
```

[file:[No Name] [+]] [Type:""] [Enc:"utf-8"] L1/24 1

図 1 my filer is working well

neovim command line

```
:lua plugin.filer.open()
```

とすると新しいウィンドウでファイルを開くか、選択したディレクトリーのファイルコンテンツがレンダリングされます。

ショートカットを登録するときは、先述の `luaeval` が使えそうですね。

neovim command line

```
:nmap wo :call<Space>luaeval('plugin.filer.open()')
```

## 5 おわりに

Vim プラグインをガリガリ書いたことないのでファイラーくらいしか思いつきませんでしたが、VimL でゴリゴリやっていくよりはだいぶ簡単だったと思います。

「コードが多く文が少ない」という指摘を前回アンケートでもらいましたが、今回も 3 ページほぼ全部コー

ドになってますね。まるで成長していない。

## 余白

余白を埋めるために『総合タワーリシチ』<sup>\*8</sup> の感想を書きます。テンポよく進んでいくギャグ漫画だな～とペラペラと読み進めていたらだんだん百合っぽくなってきて、おかしいな～おかしいな～なんて思つたらね、巨大な百合がドーンと現れて、びっくりしましたね。終わりです。みなさんも『総合タワーリシチ』を読んで、東大に合格!

---

<sup>\*8</sup> <https://www.amazon.co.jp/総合タワーリシチ-完全版-ヤングキングコミックス-あらた-伊里/dp/4785961090/>  
新規描き下ろしを含んだ完全版が発売されました。2017年はめでたい。めでたいなあ。

# 量子コンピュータなんかに絶対屈したりしない

文 編集部 みみずのひもの

## 1 初めに

RSA 「くっ、殺せ……！」

I ○ M 「げっへっへ、こんなに (4096bit) 鍵長を伸ばしやがって……もう抵抗しても無駄だア」

G ○○ gle 「量子コンピュータでこんな恥ずかしい平文までスケスケだぜ」

N ○ A 「もう二度と暗号化できないねえ……」

RSA 「ぐつ、これまでか……」

？？？「待たれよ！」

I ○ M 「誰だっ！？」

NI ○ T 「紹介しよう、新しい耐量子標準暗号方式だ。その名も……」

## 2 背景

勤勉たる WORD 読者の皆様は、「量子コンピュータ（量子計算機）」という単語をどこかで耳にした事があるだろうか。本誌編集部員 karasu 氏執筆の「量子にふれたよ！ #1 (WORD41 号)」などの記事をご覧になつた方ならば、あの量子ビット使うやべー奴じゃん www ぐらいの認識を既にお持ちかもしれない。

ざっくり説明すると現代のコンピュータが 0/1 の論理ビットで構成されるのに対して、量子コンピュータは 0/1 が確率的に決定するような量子ビット (Qbit) で構成されている。そうすると例えば古典（現代の）コンピュータは 4 bit を使って 1 つの値しか表現できないのに対し、量子コンピュータは  $2^4 = 16$  の値を同時に保持できるので、とても強力な並列計算ができるウルトラハッピーという塩梅だ。これによって量子アルゴリズムと呼ばれる専用のアルゴリズムを、量子コンピュータ上で実行することが可能となる。

そして量子コンピュータ上で実行できる最も有名なアルゴリズムと言えばそう、紛れもなく奴……ショアのアルゴリズムだ。今まで準指数時間の計算量を必要としていた素因数分解問題は、このアルゴリズムによって多項式時間で解けてしまうようになる。どれだけ因数を孕んでいる合成数でも一瞬でガバガバのガバなのだ。

さて、当然そうなると困ってくるのは、SSL 通信や PGP などで広く用いられている RSA 暗号/署名の安全性だろう。RSA 暗号方式のアルゴリズムだと復号の正当性だとかは、ちょちょいと検索すればいくらでも分かりやすい解説が登場するのでそちらを参照して頂きたい<sup>1</sup>。

重要なのはモジュラス  $n = pq$  を素数  $p, q$  に因数分解する作業が容易になつてしまふと、公開鍵  $e$  から秘密鍵  $d$  を求められるので、RSA 暗号は完全に解読されてしまうという点だ。つまり完全なる量子コンピュー

<sup>1</sup>情報セキュリティ入門の教科書、「暗号理論入門」(岡本栄司著) や、昨年出版された「暗号と情報セキュリティ (リスク工学シリーズ)」(岡本栄司、西出隆志著) などが詳しい。べつ、別に露骨な宣伝じゃないんだからねつ。

タの完成は、RSA 暗号方式そのものの死を意味している<sup>2</sup>。

さて、皆様は「ダ・ヴィンチ・コード」などで有名なダン・ブラウンの「パズル・パレス（原題:Digital Fortress）」という著作をご存知だろうか。この作品では NSA（アメリカ国家安全保障局）が秘密裏に量子コンピュータの開発に成功し、テロリストの電子メールを盗聴することでテロを未然に防いでいるという世界観が描かれている。この小説が刊行されたのは 1998 年だが、驚くべき事にそれから 15 年後、スノーデンが暴露した文書から NSA が本当に量子コンピュータ開発を進めていた事実<sup>3</sup>がうっかり判明してしまい、見事嘘から出た真となってしまった。

何も量子コンピュータ開発に興味があるのは NSA だけではない。Google や IBM などと言ったイケイケ大企業や、マサチューセッツ工科大学、中国科学技術大学、そしてここ筑波大学などといった研究機関からも、続々量子コンピュータ開発に向けての有力な研究成果がポンポン挙がっている<sup>4</sup>。ということで手遅れとなる前に、RSA の正統後継者になりうる暗号方式は早急に求められているのだ。

ちなみに情報セキュリティ (GB42101) を受講したぐらいの知識を蓄えられている方からは、「素因数分解が解けても離散対数問題に基づいた公開鍵暗号とかも色々あるんだし、そっち使えばいいじゃん」というご指摘があるかもしれない。しかし残念ながら ElGamal 暗号や楕円曲線暗号の安全性の根拠である離散対数問題<sup>5</sup>、に対しても、有効な量子アルゴリズムが既に提案されている<sup>6</sup>。ということで ElGamal 署名の応用であり標準的なデジタル署名規格である DSA (Digital Signature Algorithm) や、PS3 の認証セキュリティに用いられていることで有名な ECDSA (Elliptic Curve DSA) なども当然見直しが必要となってしまう。なかなかうまくはいかないものだ。

### 3 耐量子計算機暗号

RSA 暗号の後継を担うには、当然量子コンピュータでも解読できないような暗号方式でなくてはならない。そのような暗号方式の事を「耐量子計算機暗号」もしくは「耐量子暗号」と呼ぶ。さらに英語だと "Post-Quantum Cryptography" なのでポスト量子暗号という呼び方もあり、今のところ名称は統一感が無くガバガバのプレブレである。耐性を持ちたいのは量子計算機なので、個人的には耐量子計算機暗号が一番しっくりくると思う

---

<sup>2</sup> 実際 RSA を解読するのにそれなりの Qbit を保有する量子計算機が必要なため、Terabyte 単位の素数を用いる RSA なら解読されないので？ という研究の方向性もあるとか（ただし現状では暗号化・復号に 5 日間かかる）。詳細は下記のページ "Why Quantum Computers Might Not Break Cryptography" より。

<https://www.quantamagazine.org/why-quantum-computers-might-not-break-cryptography/>

<sup>3</sup> 「米 NSA が量子コンピューター開発中か、暗号解読用と米紙」

<http://www.afpbb.com/articles/-/3005926>

<sup>4</sup> 量子コンピュータにはゲート方式（素因数分解、機械学習などの汎用的な計算処理向け）、量子アニーリング方式（最適化問題、サンプリングのみに限定された計算処理向け）という 2 つの方法が存在しているが、量子アニーリング方式と比較するとゲート方式の実用化にはまだ時間がかかるとのことらしい。現代の量子コンピュータ開発ブームはどちらかと言うとアニーリングがもてはやされてる（らしい）ので、量子コンピュータ実現と素因数分解可能は少し切り分けて考える必要がある。詳しくは下記のリンクに。  
<http://ascii.jp/elem/000/001/451/1451932/>

<sup>5</sup> ElGamal は素数の剩余環上、楕円曲線暗号は楕円曲線上における離散対数問題の困難性を、それぞれ安全性の根拠としている。

<sup>6</sup> 「量子計算機の研究動向に関する調査」調査報告書 Sec.4.3 「暗号解読に関するアルゴリズム」に具体的な手法が書かれている。  
<https://www.ipa.go.jp/security/enc/quantumcomputers/contents/contents.html>

んですがそれはどうなんですかね……。さらにややこしいことに、後述する「量子暗号」とは別物なので注意。とりあえず本記事での表記は大量死、じゃなくて耐量子計算機暗号で統一する。

そんな訳で NIST（アメリカ国立標準技術研究所）は、2016 年から耐量子計算機暗号の募集を始めているそうだ<sup>7</sup>。世は大暗号時代。これから地下闘技場編の如く量子計算機の猛攻に耐えうる世界各地の猛者たちが NIST に集い、そして血で血を洗う闘争の後、その頂点のみが標準暗号方式となりうるのだ。

ということで読者の皆様方の期待も否応なしに高まってきた頃合いだろう。本記事では次世代を担う可能性が高い耐量子計算機暗号達の一部を紹介しよう。

### 3.1 格子暗号

編集者の一推し暗号。これを推しておけばまず間違いはないので、ジャンジャン推してほしい。

数学の「格子」と呼ばれる空間上では、定義はシンプルながら解くのが非常に難しいとされている問題がある。格子暗号とは文字通り、こういった格子の問題を安全性の根拠とした暗号方式のことである。格子の定義は以下の通り。

格子 (Lattice) -

次元数  $n$ 、基底数  $m$  からなる互いに線形独立な格子基底  $B \in \mathbb{Z}^{m \times n}$  を用いて、格子上の点の集合  $L(B)$  は

$$L(B) = \{ Bx \mid x \in \mathbb{Z}^n \}$$

で表せる。

一番簡単な格子の例として、2mm×2mm の方眼紙のようなものを想像してもらうと分かりやすい。方眼紙は  $(2, 0)^T, (0, 2)^T$  という 2 つの格子基底から構成される格子なので、例えば座標  $x = (2, 3)^T$  に対応する格子上の点は

$$Bx = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

となり、(4mm, 6mm) の位置に相当する。

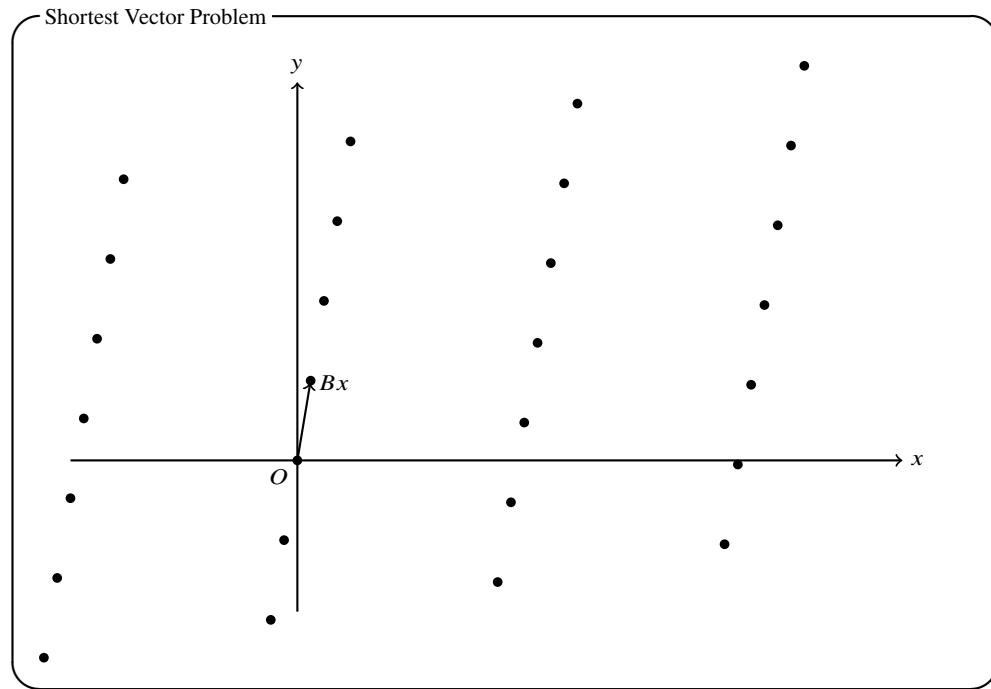
なお一般に「格子」と聞くと格子基底同士が直交、つまり正方形を敷き詰めた形状のイメージが先行して思い浮かぶかもしれない。しかし定義上互いに線形独立な基底であれば格子となりうるので、例えば平行四辺形を敷き詰めたような形状でも格子の 1 つである。格子はフリーダム、自由な形状で構成されていていいんだ！ ということを念頭に置きつつ今後は読み進めて頂きたい。

さてこのような格子  $B$  を定義した時、以下のような 2 つの問題が考えられる。なお以降ノルム  $\|\cdot\|$  はユークリッドノルムとする。

<sup>7</sup> 「量子コンピュータの実用化見据え、求められるセキュリティ-NIST が耐量子暗号の標準策定へ」  
<https://japan.zdnet.com/article/35087380/>

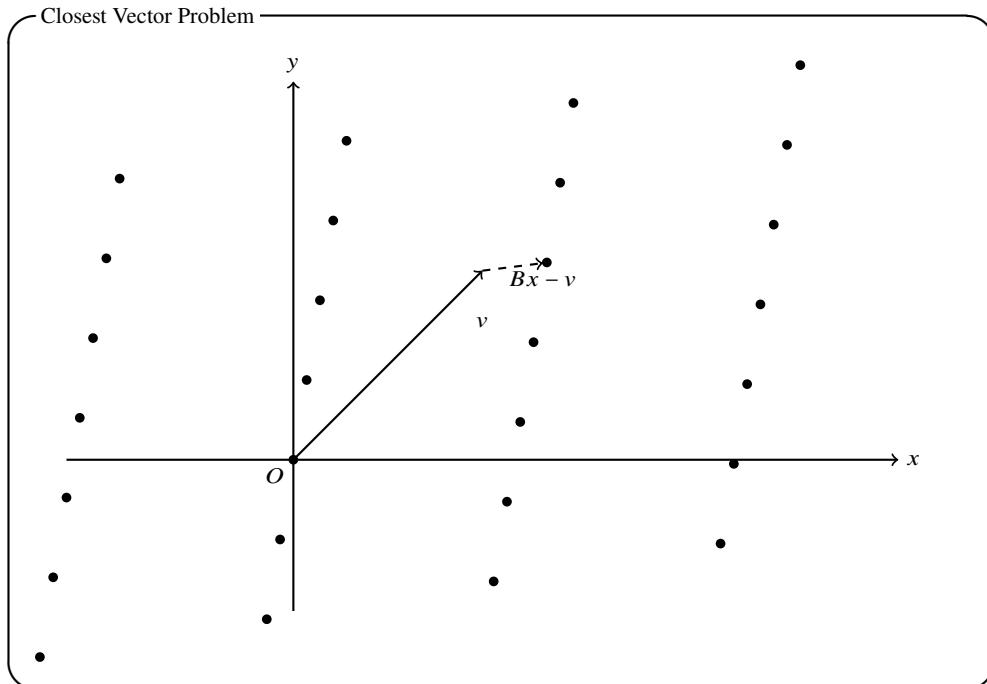
- 最短ベクトル問題 (Shortest Vector Problem, SVP)

原点  $O$  から距離  $\|Bx\|$  が最短となるような非零ベクトル  $x$  を求めよ。



- 最近ベクトル問題 (Closest Vector Problem, CVP)

あるベクトル  $v$  が与えられた時、任意の格子点との距離  $\|Bx - v\|$  が最短となるような非零ベクトル  $x$  を求めよ。



先程の方眼紙の例でイメージすると、一見そこまで難しくない問題に見える。しかし我々の理解が及ぶのはあくまで二次元、三次元レベルでのお話だ。十分な次元数とユークリッドノルムを持つ複雑な SVP は NP-困難であり、さらに CVP は SVP を一般化したより困難な問題であることが既に示されている（CVP が解ける → SVP が解ける）。

事実 SVP とその近似問題である Approx-SVP の解読を試みるプロジェクト「SVP CHALLENGE<sup>8</sup>」によると、2017 年 6 月現時点では SVP の世界記録は、次元数が 150 でユークリッドノルムが 3220 である。逆に言うと次元数が 200 か 300 ぐらいあるような複雑な格子を用いれば、ベクトル  $x$  を求める作業は現代のスパコン、さらに言えば量子コンピュータを用いても困難であるということになる<sup>9</sup>。そしてこの困難性こそが、まさに格子暗号の安全性の根拠に相当するものだ。

<sup>8</sup><https://www.latticechallenge.org/svp-challenge/index.php>

<sup>9</sup>現状はあくまで現実的な解くアルゴリズムが発見されていないというだけで、RSA のように新しい計算機・アルゴリズムさえ見つかればあっさり解かれる可能性を秘めている点に注意。

SVP HALL OF FAME					
Position	Dimension	Index	Seed	Contestant	Solution
1	130	131	0	2912 Shang-Yi Yang	vec ...
2	130	262	0	3000 Jean-Christophe Deneuville	vec ...
3	130	262	0	3004 Kenji KASHIWABARA and Tadanori TERUYA	vec ...
4	128	255	0	2924 Kenji KASHIWABARA and Tadanori TERUYA	vec ...
5	128	256	0	2959 Tsukasa Ishiguro, Shinzaku Kiyomoto, Yutaka Miyake, Tsuyoshi Takagi	vec ...

APPROX-SVP HALL OF FAME					
Position	Dimension	Index	Seed	Contestant	Solution
1	652	653	0	626850 Yuntao Wang; Yoshinori Aono; Takuuya Hayashi; Jintai Ding; Tsuyoshi Takagi	vec ...
2	652	653	0	626936 Yuntao Wang; Yoshinori Aono; Takuuya Hayashi; Tsuyoshi Takagi	vec ...
3	652	653	0	661210 Jean-Christophe Deneuville	vec ...
4	652	653	0	661349 Takuuya Hayashi, Tsuyoshi Takagi	vec ...
5	600	601	0	542883 Jean-Christophe Deneuville	vec ...

ところで。世界中の研究者が互いに SHINOGI を削っている SVP CHALLENGE だが、SVP や LWE(後述)のチャレンジランキングに Tadanori TERUYA という名前を見ることができる。こう見ると「成る程、日本人の研究者も積極的にやつとるんだなあ」ぐらいの認識かと思われるが、実はそれどころの話ではない。照屋唯紀氏はここ筑波大学情報学類（旧名）を卒業し、筑波大学大学院の博士号を取得された現在産総研の研究員であり、さらには WORD 編集部のOB だったりする。世間は狭い、あまりにも狭いのだ……。他にもランキングからは日本人研究者の名前がちらほら見受けられる。

さて、話を元に戻そう。古来から暗号学者は素因数分解問題然り離散対数問題然り、「計算機を使っても解くのが難しい」問題を暗号へと転用してきた経緯がある。次章では格子上の問題を困難性の根拠とした暗号方式の1つであり、Gentry らが2013年に発表した方式である GSW13について説明しよう。

### 3.2 GSW13（格子暗号の一種）

確かに SVP の問題そのものは NP-困難ではあるものの、これを暗号として安全性証明をつけるのは難しい。そこで SVP をそのまま利用するのではなく少し簡単にした Approx-SVP とその Approx-SVP より難しい LWE という計算問題を導入し、LWE を使って暗号方式を構成する事について考える。

ではまず Approx-SVP について考える。

#### Approx-SVP —

格子基底  $B \in \mathbb{Z}^{m \times n}$  を先述のように定義する。この時  $\|e\| < \beta$  を満たすような任意のベクトル  $e \in \mathbb{Z}^m$  を用いて

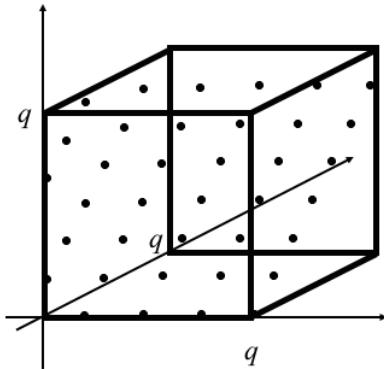
$$Bx^T = e$$

を満たすようなベクトル  $x \in \mathbb{Z}^n$  を求めよ<sup>\*a</sup>。

<sup>\*a</sup>  $\beta$  には最短ベクトルノルムの定数・実数倍の値が与えられる

つまり SVP は厳密な最短距離となる格子点を求める問題だったのでに対し、Approx-SVP ではある程度小さい距離となる格子点なら OK という若干ゆるゆるな問題だ。 $\beta$  が最短ベクトルのノルムと同一ならば、Approx-SVP は SVP に等しくなる。当然ながら  $\beta$  の値の取り方によって Approx-SVP の難しさは大きく変化

するが、 $\beta$  が定数倍程度ならば量子コンピュータでも十分に解読が難しいほどの強度を保つのでご安心頂きたい。



さて、格子は定義上無限の空間に広がるが、構造は基本的に繰り返しなのでモジュラス  $q$  の剩余環上でも同じ事を考えられる。このような格子を特に  $q$ -ary 格子と呼ぶ。例として三次元空間上の  $q$ -ary 格子だと、左図のように  $q \times q \times q$  の立方体内部に全ての格子点がうようよ点在している状態をイメージして頂くと分かりやすい。

この  $q$ -ary 格子を用いて定義されるのが、以下の Learning With Errors (LWE) の計算問題である。

#### Learning With Errors

$n$  を次元数、 $m$  をサンプル数、 $q$  を十分に大きいモジュラス、 $A' \in \mathbb{Z}_q^{n-1 \times m}$  を  $q$ -ary 格子基底行列とおく。この時ランダムに選択した  $s \in \mathbb{Z}_q^{n-1}$  と、 $\|e\| < \beta$  を満たすような  $q$  に比べて十分に小さいベクトル  $e \in \mathbb{Z}_q^m$  を用いて  $b$  を、以下のように定義する。

$$b \leftarrow sA' - e$$

この時サンプル  $(A', b)$  を入力として与えられた時、 $s$  を求める問題を Learning With Errors(LWE) の計算問題と呼ぶ。また  $t = (s, 1)$ ,  $A = (A', -b)$  とおくと、

$$tA = sA' - b = e$$

という関係式に置き換えられる。

この置き換えによって Learning With Errors は、行列  $A$  を与えられた時に積が  $e$  となるようなベクトル  $t$  を求める問題となる。

Approx-SVP と比較すると、LWE でパラメータを取ってくる空間が  $\mathbb{Z}$  から  $\mathbb{Z}_q$  へと変化している点にご注目頂きたい。Approx-SVP では原点から極端に遠いような格子点はそもそも最短の候補から省かれてしまうが、LWE では  $q$  をモジュラスとして循環するので、様々な格子点が最短の可能性を内在している。要する

に解として取りうる候補は LWE 決定問題の方が多くなるという事が分かる。

ということでここまで露骨な誘導をすればもはやお察し頂けただろう……。

そう、来る 2005 年！ Regev によって LWE 決定問題は Approx-SVP と少なくとも同程度に困難であることが示されたのだ<sup>10</sup>！ つまり難しさのイメージとしては (Approx-SVP の困難性)  $\leq$  (LWE の困難性)  $<$  (SVP の困難性) となる。

「 $tA = e$  という式について、 $t, e$  から  $A$  を計算するのは簡単だが、 $A$  から  $t, e$  を求めるのは難しい」

これこそが格子暗号業界における衝撃的なブレイクスルーであり、格子暗号の歴史上でも重要なターニングポイントとなった出来事だ。

実のところ格子暗号には他にも GGH, NTRU など言った方式が名を連ねているが、それらと比較すると安全性が高く（クリティカルな攻撃方法があまり見つかっていない）、かつ  $tA = e$  という数学的に扱いやすい式で構成されているという点から、プロトコルが構成しやすいので暗号学者の間でもウケがよい。こうして現在に至るまで、LWE を用いた暗号方式や一方向性関数は特に数多く提案されている。

ということで前置きは長くなってしまったが、当初の目的である GSW13 の構成に話を戻そう。

## 暗号化

より厳密な話をするためには細かい定義の話が次から次へと雪崩のように押し寄せてくるため、今回はざっくりとした説明にとどまらせて頂く。

まず上述の LWE 仮定から生成されるベクトル  $t$  を秘密鍵、行列  $A$  を公開鍵とおく。ここで RSA 暗号方式で公開鍵  $e$  から秘密鍵  $d$  を求める事が困難であるのと同様、LWE 仮定の困難性に基づくと公開鍵  $A$  から秘密鍵  $t$  を求めるのは困難である。

$\mu \in \{0, 1\}$  を平文（秘密のメッセージ）、 $R \in \{0, 1\}_q^{m \times m}$  を一様にランダムなビット行列、Gadget vector と呼ばれる特殊なベクトルを  $g = (1, 2, 2^2, \dots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell$  としてそれぞれおくと、GSW13 の暗号文  $C$  は

$$C \leftarrow AR + \mu \cdot (I_n \otimes g)$$

という形で構成される（ $I_N$  は単位行列、 $\otimes$  はテンソル積）。

読んでいる方もいよいよをもってなんのこっちゃとなってくる頃合いかと思われる所以、数式が苦手な方は  $AR$  の項を「モザイク」、 $\mu \cdot (I_N \otimes g)$  の項を秘密の「お宝画像」としてイメージしていただきたい。この状態から純粋に暗号文  $C$  から平文  $\mu$ （お宝画像）を解読しようと試みても、ノイズ項（モザイク）が邪魔しているため不可能であるお気持ちもご理解頂けるだろう。

## 復号

復号の仕組みはシンプルで、秘密鍵  $t$ （モザイク除去機）を  $C$  の左から掛けることで平文の情報が得られる。 $M = tC$  とおくと

---

<sup>10</sup>厳密には  $SVP_\gamma, SIVP_\gamma$  と呼ばれる問題の困難性に帰着する。

$$\begin{aligned}
M &= tC \\
&= tAR + \mu t \cdot (I_N \otimes g) \\
&= eR + \mu \cdot (t \otimes g) \\
&\approx \mu \cdot (t \otimes g)
\end{aligned}$$

という計算で、ノイズ項  $eR$  を近似計算によって擬似的に消去している様子がお分かり頂けただろうか。しかしモザイク除去機を用いても鮮明な映像が得られないのと同様に、完全な形で元のお宝情報が手に入るわけがない。

そのため続けて  $\mu$  の情報を抽出する。 $t \cdot \bar{t} = q/2$  を満たすような  $\bar{t}$  を使って、 $M$  の右から  $(\bar{t} \otimes g^{-1})$  をかけてやると

$$\begin{aligned}
M \cdot (\bar{t} \otimes g^{-1}) &\approx \mu \cdot (t \cdot \bar{t} \otimes g \cdot g^{-1}) \\
&= \mu \cdot \frac{q}{2}
\end{aligned}$$

最後に  $M \cdot (\bar{t} \otimes g^{-1})$  に対して丸め計算（四捨五入）をすることで、結果が  $q/2$  ならば  $\mu = 1$ 、0 ならば  $\mu = 0$  となるので、晴れて無事に復号されるというわけだ。めでたしめでたし。

### 問題点

ということで無事暗号へと転用する事が出来たわけだが、「おや？」と頭上に疑問符が浮かび上がった賢明な読者様も数多くいらっしゃるだろう。それは恐らくここまで手間暇かけておいてたったの 1bit しか暗号化出来とらんのかいという疑問だが、まさにその通りである。

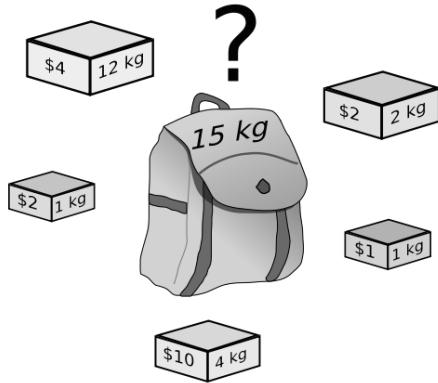
LWE ベースの暗号方式最大の問題はそこにあり、平文 1bit を暗号化すると暗号文は  $n \times m$ （式にもよるが大体  $200 \times 1000$  ぐらい）という馬鹿でかい行列に膨れ上がってしまうのだ。では  $k$  bit の平文を暗号化するためにはどうするかと聞かれたら、そらしうがないので、泣きながら  $k$  個の暗号文を使うのだ……。この効率の悪さは RSA 暗号方式（2048bit を暗号化して 2048bit の暗号文）ElGamal 暗号方式（2048bit を暗号化して 2048bit の暗号文が 2つ）などと比較すると、圧倒的である事がお分かり頂けるだろう。

しかし理論的観点から見れば実現できたので何の問題もない（強調）。実用化に向けては HPCS や並列計算など、また別の研究に頑張ってもらうことになる。

ところでこれは耐量子計算機の話とは別件だが、GSW13 の暗号方式は「完全準同型暗号」という性質を持っている。これは簡単に言うと暗号文同士の加算で平文が加算できたり、暗号文同士の乗算で平文が乗算できたりという超便利性質だ。この部分を語り始めると無限に気持ち良い話が始まってしまうのだが、流石に紙面上のキリがなくなるので、興味のある方はぜひ暗号情報セキュリティ研究室までお越し頂きたい……。

### 3.3 量子ナップサック暗号

少年漫画の王道と言えば、一度死んだかと思われた味方が後半に復活し、主人公のピンチに駆けつける展開だろう。イエス・アイ・アム。量子ナップサック暗号はそんな立ち回りだ。



まず原型となる Merkle-Hellman ナップサック暗号(1978 年)は、その名の通りナップサック問題を解読困難の根拠においていた暗号方式だ。ナップサック問題と言えば、「最大荷重  $C$  のナップサックが  $I$  つと、 $i$  番目が価値  $p_i$  重さ  $c_i$  を持つような品物  $a_i$  が  $a_1, \dots, a_n$  まである。ナップサックの最大荷重を超えない範囲でナップサック内の価値を最大化するにはどの組み合わせで品物を詰め込めばよいか?」という線形計画問題で、問題の理解し易さ、コーディングのしやすさ、ヒューリスティック手法の適用しやすさなどなど、数多の萌え要素を含んでいることでも有名だ。特に  $p_i = c_i$  である時これは部分和問題と呼ばれ、NP 完全であることが広く知られている。

<https://commons.wikimedia.org/wiki/File:Knapsack.svg>

それではこの暗号方式について、実例を踏まえてざっくり説明しよう。

- 準備

超増加列<sup>11</sup>の品物の価値のリスト  $A = \{ A_1, A_2, \dots, A_n \}$  と、秘密鍵  $r$  をランダムに生成する。また演算は全て  $M$  を法とするモジュロ演算で行われる。

ここでは  $A = \{ 1, 2, 4, 8, 16 \}, r = 3, M = 19$  とする。

- 平文

$n$  個の荷物を選択する/しないのビット列  $m \in \{ 0, 1 \}^n$

ここでは  $m = \{ 0, 1, 1, 0, 0 \}$  とする。

- 公開鍵

$A$  の各要素をそれぞれ  $r$  倍してからソートしたリスト  $A'$

ここでは  $r \cdot A = 3 \cdot \{ 1, 2, 4, 8, 16 \} \equiv \{ 3, 6, 12, 5, 10 \} \pmod{19}$  となるので  $A' = \{ 3, 5, 6, 10, 12 \}$  となる。

---

<sup>11</sup> $A_n > \sum_{k=1}^{n-1} A_k$  を満たすような列。一番簡単な例としては  $1, 2, 4, 8, \dots$  みたいな  $2$  のべき乗の列。

- 暗号文

$A'$  の中で平文のように荷物を選択した時の価値の総和  $C$

ここでは  $C = 5 + 6 = 11$  とする。

- 復号

$C' = r^{-1} \cdot C$  をナップサックの容量として、リスト  $A$  の部分和問題を解き<sup>12</sup>、対応する要素を  $A'$  から選択したもののが復号結果となる。

ここでは  $C' = 13 \times 11 \equiv 10$  となるので、リスト  $A$  の部分和問題の解は  $\{ 0, 1, 0, 1, 0 \}$  となる。すなわち  $A'$  でこれに対応する  $\{ 0, 1, 1, 0, 0 \}$  が復号結果となる。

ということでこのナップサック暗号の安全性の根拠は、部分和問題に相当する。先述のとおりこれは NP 完全問題なので安心……かと思っていたのもつかの間、1982 年、あの RSA 三銃士の 1 人、アディ・シャミアによって攻撃法が発見されてしまった。これはリスト  $A$  が超増加列である性質の脆弱性を突いた攻撃であり、部分和問題とは直接渡り合はずとも解読する事ができる。流石シャミア博士、俺の作った公開鍵暗号方式以外は認めんという頑なな意志が窺えて気持ちが良い<sup>13</sup>。

というわけでその後ナップサック暗号には幾つか派生系が誕生したものの、次々とどれも解読可能であることが示されてしまったらしい。そのまま人目に付くこともなくひっそりと息絶えたナップサック暗号であつた……が、彼はまさかの形で復活する。そう、それこそが量子ナップサック暗号だ。

これはその名に冠する通り、本来攻撃者側であるはずの量子コンピュータを用いて鍵生成することによって<sup>14</sup>、既存の攻撃にはもちろんのこと量子計算機にも耐性を持った暗号方式となったのである。有名な例としては、岡本龍明教授らの提案による OTU2000 などがある<sup>15</sup>。

「敵の力を使って自身を強化する」という、まさに仮面ライダー 1 号から超人サイバー Z に至るまでヒーロー達が戦いの中で取得した有名メソッドは、なんと遠くかけ離れた暗号分野でも応用されていたのだ。量子コンピュータを倒すには量子コンピュータをぶつければいい。シンプルな話だ。臥薪嘗胆の思いで煮え湯を飲まされてきたナップサック問題くんは、魑魅魍魎が蔓延る新時代においてようやくスポットライトを浴びる事ができるのかもしれない……。

---

<sup>12</sup> イメージとしてはリスト  $A$  は「激簡単」ナップサック問題、リスト  $A'$  は普通のナップサック問題なので、秘密鍵を知っている人間は簡単に復号できるという気持ち。

<sup>13</sup> シャミア博士といえば先日 2017 年日本国際賞を受賞されたそうです。さすシャミ！（流石ですシャミア様！）  
<http://www.itmedia.co.jp/enterprise/articles/1702/02/news014.html>

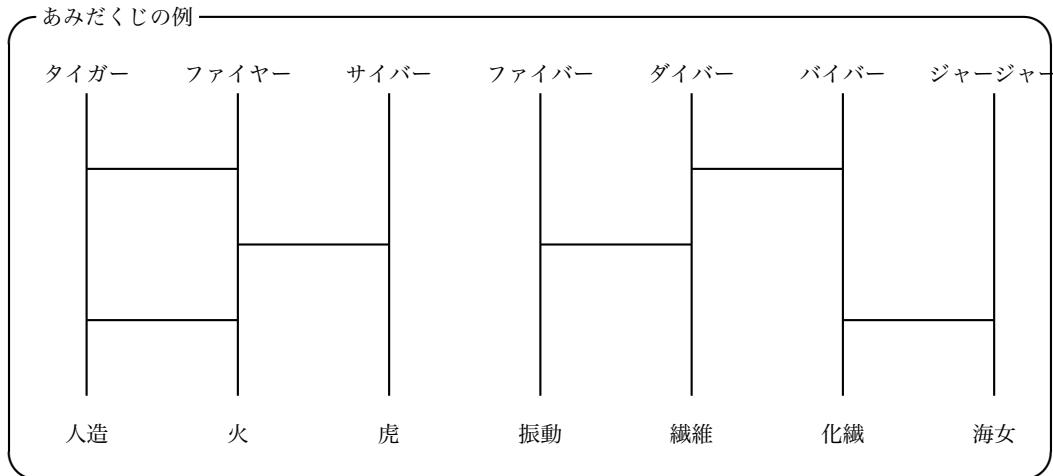
<sup>14</sup> 量子コンピュータを使うと離散対数問題を解くことができる。それを逆手に取って OTU2000 の鍵生成アルゴリズムでは、離散対数を量子コンピュータで計算するステップが導入されている。

<sup>15</sup> 「量子公開鍵暗号について」  
<http://staff.miayko-u.ac.jp/~taya/sendaiNC/2005/report/miyazawa-uchiyama-okamoto.pdf>

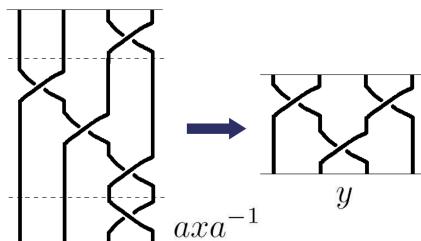
### 3.4 組み紐群暗号

組み紐（ブレイド）群を使うというちょっと不思議な暗号方式も、耐量子計算機暗号として名乗りを挙げているのでご紹介しよう。

$n$  ブレイドとは  $n$  本の紐で構成される群である。紐の両端にはそれぞれ入力と出力があり、また紐と紐の位置は途中で入れ替わってもよい。最も身近な例として、あみだくじはまさにブレイドそのものである。



2つの  $n$  ブレイドはそれぞれの出力と入力をつなぎ合わせる事で、合成が可能である。任意のブレイド  $a$  には逆元  $a^{-1}$  (ひっくり返したあみだくじ)、そして単位元のブレイド  $e$  (横線を引かないあみだくじ) も存在する。ということで  $n$  ブレイドは皆様お馴染みの、群を成すことが分かる。



次は困難性の根拠である共役だ。任意のブレイド  $(x, y)$  について  $ya = ax$  を満たすような  $a$  が存在するならば、 $(x, y)$  は共役であると定義される。ここでブレイド群が十分に複雑ならば、共役であるような  $(x, y)$  から  $a$  を探索する問題（共役探索問題）は、量子コンピュータを用いても解くことが困難であると予想されている。共役なブレイドの関係については、図でイメージしてみると分かりやすい<sup>16</sup>。

共役なブレイドの図

<sup>13</sup>「ブレイド群を利用した公開鍵暗号系の実装と性能評価実験」  
<http://www.tani.cs.chs.nihon-u.ac.jp/g-2005/yuko/braid.pdf>

$(x, a)$  から  $y$  を求めるのは容易いが、 $(x, y)$  から  $a$  を求めるのが困難であるという性質から、離散対数問題の性質にどことなく似ているという気持ちがお分かり頂けるだろうか。この共役探索問題の困難性、もしくはそれを応用した一方向性関数の E-Multiplication などをベースとして、現在までに様々な暗号方式が提案されている。

ここでは 2000 年初出という古い提案手法だが、組み紐を用いた暗号方式を 1 つ紹介しよう<sup>17</sup>。

- 準備

十分に複雑なブレイド群を  $B_{l+r}$  とおく。ブレイドは左側の  $l$  本の紐で構成される  $LB_l$  と、右側の  $r$  本の紐で構成される  $RB_r$  という 2 つの部分群で構成される。この時任意の  $a \in LB_l$  と  $b \in RB_r$  は可換である。またブレイドを入力として受け取り  $k$  bit の乱数を返すようなハッシュ関数  $H : B_{l+r} \rightarrow \{0, 1\}^k$  を用意する。

- 鍵生成

Bob は秘密鍵  $a \in LB_l$  を用いてランダムに選択した  $x \in B_{l+r}$  から  $y = axa^{-1}$  を計算し、 $(x, y)$  を公開鍵として公開する。この時公開鍵のペアから秘密鍵を求める問題は、共役探索問題に相当するため困難である。

- 暗号化

Alice はメッセージ  $m \in \{0, 1\}$  を以下の手順で暗号化する。

1.  $b \in RB_r$  をランダムに選択する。
2.  $c = bxb^{-1}$  を計算する。
3.  $d = H(byb^{-1}) \oplus m$  を計算する。
4.  $(c, d)$  を暗号文として Bob に送信する。

- 復号

Alice は  $m' = H(aca^{-1}) \oplus d$  を計算することで平文  $m'$  が得られる。

実際に  $m' = m$  を導くことで正しく復号されるかを確認してみよう。ポイントは  $a, b$  が可換というところだ。

---

<sup>17</sup>画像は以下より拝借。

[http://www.tani.cs.chs.nihon-u.ac.jp/g-2005/yuko/braid\\_sxi.pdf](http://www.tani.cs.chs.nihon-u.ac.jp/g-2005/yuko/braid_sxi.pdf)

$$\begin{aligned}m' &= H(aca^{-1}) \oplus d \\&= H(abxb^{-1}a^{-1}) \oplus H(byb^{-1}) \oplus m \\&= H(baxa^{-1}b^{-1}) \oplus H(byb^{-1}) \oplus m \\&= H(byb^{-1}) \oplus H(byb^{-1}) \oplus m \\&= m\end{aligned}$$

### 3.5 量子暗号

それでは大本命、量子暗号の説明に移るとしよう。量子暗号が他の暗号方式に対して大きく差をつけている点は、これが計算量的安全性（解読に時間がかかる安全性）ではなく、情報理論的安全性（無限に時間を費やしても解読できない安全性）に基づいていることだろう。

これだけ聞くと何故そんな夢のような技術が可能なのだろうか、と訝しまれる方も多いかとは思われるが、実のところ種を明かせば大したことはない。量子暗号はシャノンによって情報理論的に絶対安全であることが証明された暗号方式、ワンタイムパッド（バーナム暗号）を用いているからだ。

量子暗号は「量子鍵配達」と「ワンタイムパッド」の2つで構成されている。ここでは Bennet らによって1984年に提案された、最もシンプルでポピュラーな量子鍵配達プロトコル「BB84」を紹介しよう。

#### 量子鍵配達

1. Alice は Bob にランダムな光子列を量子通信路で送信する。光子は直線偏光である「縦(↑)」「横(→)」、円偏光である「右回り(↗)」「左回り(↖)」という4つの状態のいずれかを保持しているものとする。
2. Bob は得られた光子1つ1つに対して、検出器 A と検出器 B のどちらで検出するかをランダムに選択する。この時検出器 A は直線偏光検出系（縦か横かを判別できる）、検出器 B は円偏光検出系（右回りか左回りかを判別できる）の属性を備えている。
3. 以下の表に従ってビット列を出力する（ただし ↖ は正しく検出されていない結果とする）。

	↑	→	↗	↖
検出器 A	0	1	⊥	⊥
検出器 B	⊥	⊥	0	1

4. Bob はどの時刻にどちらの検出器を用いたかの情報を  $(ABBABA\dots)$  を Alice に送信する（これは公衆通信でよい）。
5. Alice はどの時刻に正しく検出できているかの情報を Bob に送信する（同じく公衆通信でよい）。この時一部のデータを犠牲にし、同じビット列を共有できているかを確認する。確認に失敗した場合は盗聴者が存在するものとして破棄する。

- 
6. Alice, Bob は得られたビットの乱数列を共通鍵として共有する。

上のプロトコルを用いて鍵配達する様子を下の表で示そう。

時刻	0	1	2	3	4	5	6
Alice の生成した光子	↑	↑	～	～	～	～	→
Bob の使用した検出器	A	A	B	B	A	B	B
Alice が答える検出器の正誤	○	○	○	○	×	○	×
Alice, Bob で共有されるビット列	0	0	0	1		0	

## ワンタイムパッド

Alice と Bob の間で先程共有した共通鍵を  $r$  とおく。

1. Alice は平文  $m$  を Bob へと送信するため、暗号文  $c = m \otimes r$  を計算する。ここでの  $\otimes$  はビット毎の排他的論理和を意味し、 $m$  のビット長が  $r$  のビット長に満たなかったり長過ぎる場合はパディングしたり分割したりして何とかする。
2. Alice は Bob に  $c$  を送信する。
3. Bob は  $m' = c \otimes r$  を計算し、秘密のメッセージを取得する。

この方式はバーナム暗号方式と呼ばれ、特に鍵を 1 回しか使用しない（使い捨て）ようなプロトコルをワンタイムパッドと呼ぶ。これは数ある暗号方式の中で唯一情報理論的安全性が数学的に証明されており、歴史上唯一無二にして正真正銘最強の暗号である。ただし鍵を平文と同じ長さ容易しなければならない、毎回鍵を新しく発行しなければならないと制約も多いため、その手間を惜しまない銀行・ホットラインなど的一部でしか利用されていないのが現状だ。

そこで登場するのが、量子鍵配達の圧倒的データ転送力である。短時間で大量の光子を転送可能な量子鍵配達は、まさにバーナム暗号のようなリソースを食う暗号方式にピッタリ噛み合い、量子暗号方式は産声を上げたのだ。量子暗号ではこれ以外にも、伝送中のノイズを訂正する量子誤り訂正符号、量子もつれの性質を利用した量子中継などの技術を複合させることによって構成される。

## 安全性

ということで、暗号文自体の安全性は無事バーナム暗号に依存しているから安心！ と相成った。だがちょっと待て、生のままデータを送信している鍵の方は安全なのか？ と意義を唱える貴殿はとても鋭い。これこそが量子暗号における安全性の肝となる部分だ。

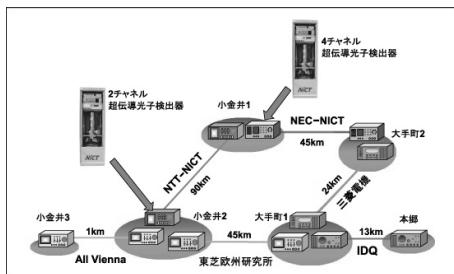
では盗聴者 Eve が量子通信路を盗聴する時の気持ちになって考えてみよう。まず Alice から何かしらの光子列が得られる。ここで Eve は「1 つの光子に対して 2 つの測定器で同時に測定する」という事を実施しようとするが、それはできないお約束となっている。何故ならある粒子を同時かつ正確に測定する事はできないという、量子力学の大原則「不確定性原理」が存在するからだ。そのため Eve は各 Qbit に対して、測定器 A を用いるか測定器 B を用いるかを 1/2 で選択する決断に迫られる。

そこで Eve は閃く。このまま素知らぬ顔で Bob へと光子列を素通しすれば、正しい観測機の情報が後で Alice から Bob へと流れてくるのではないだろうか？ Eve は早速光子列をコピーしようとしたが出来ない。これは同じ状態の量子を複製することは不可能であるという、量子力学の原則「複製不可能定理」が存在するからだ。

ではもはやヤケクソで、Eve が Alice から送信されたデータを受信した後に、めちゃくちゃな光子列を Bob に送ったらどうなるのか。これも問題ない。共有されたデータの一部を犠牲にする最後の確認フェイズで、Alice と Bob で共有されているデータが一致しないならば、盗聴されているものとしてそれを破棄してやり直すからだ。

このようにして BB84 では安全性を確保している。

## 実用化



この暗号方式が他の方式と比較して一步進んでいる点は、何と言っても日本の電気電子メーカーの注目度と、実用に向けての盛んな取り組みだろう。2010 年には既に NEC, 三菱電機, NTT そして NICT という名だたる企業・組織の協力により、量子暗号ネットワークでの長期運用実験が行われている。

さらに実際に盗聴することで、通信経路に耐性がある事を確認できたとの報告が挙げられている<sup>18</sup>。既に東京の地下にこういった巨大量子暗号通信ネットワークが形成されていたという事実は、知らなかつた人も多いだろう。

また量子暗号実現の妨げとなっていた技術的困難性に対して、次々と解決策が提案されているのも絶賛追い風となっている。

例えば今まで量子暗号を実現するために、大容量転送可能な光ファイバを新たに設置する必要があるとされていたが、既存の光ファイバを用いて構成する方法が NICT から提案された。これにより導入に向けて大幅なコスト削減が望まれるとされている。

他にも量子通信ではファイバの光損失の影響を受けるため、従来の理論上では 400km 圏内までしか通信圏を伸ばすことはできなかった。しかし NTT 研究所から発表された新方式によれば、これを 2 倍の 800km 圏内までカバーできるようになるそうだ<sup>19</sup>。これは日本国内の主要都市間同士でネットワークを構築できる距

<sup>18</sup>実用化まであと一步「量子暗号」ネットワークの研究  
[http://www.nict.go.jp/publication/NICT-News/1102/NICT\\_NEWS\\_1102\\_J.pdf](http://www.nict.go.jp/publication/NICT-News/1102/NICT_NEWS_1102_J.pdf)

<sup>19</sup>「盗聴不可能な量子暗号の通信距離を 2 倍にする新方式を提唱」  
<http://www.ntt.co.jp/news2015/1512/151216a.html>

離にあたる。この通信可能圏も研究の推進によって拡大し、じきに国家間での量子通信が可能となるだろう。

現状の量子暗号はまだ研究レベルでの開発であり、これが一般ユーザーの手に渡るまであと 10 年は必要だろうと NICT では見通している。とはいっても現状の耐量子計算機暗号方式の中では、十分に実用的であり有力な暗号方式の 1 つに当たるだろう。

## 4 終わりに

ということいかがだっただろうか。今回は量子計算機に耐性のある暗号方式の一部を紹介したが、他にも多変数公開鍵暗号、誤り訂正符号を用いた暗号方式など、次世代を担う可能性のある暗号は無数に存在する。

ここで重要なのは整数論・量子力学・システム最適化など、多様な分野で培われた科学技術が暗号技術もしくは解読への応用を可能としている点だ。応用が難しいとされていた初等整数論から RSA 暗号方式が生まれ、逆に暗号エニグマを解読するためにチューリングは計算機を生み出した。これこそが暗号という研究分野が持つ柔軟性の高さでもあり、色々な分野に首を突っ込んで美味しいとこ取りができる面白みでもある。

ひょっとしたら貴方が現在研究している内容も、実は暗号への応用ができるかもしれない……。そんな閃きが訪れたら、ぜひともジャンジャンセレンディピティデルノザウルスして頂きたい。暗号情報セキュリティ研究室は君の活躍を待っている。

## 5 参考文献（脚注に乗せられなかったもの）

- 「格子問題等の困難性に関する調査」

[https://www.cryptrec.go.jp/estimation/techrep\\_id2404.pdf](https://www.cryptrec.go.jp/estimation/techrep_id2404.pdf)

- 「次世代暗号・認証方式の研究・開発に関する調査報告—量子暗号に関する調査・研究報告書—」

<https://www.ipa.go.jp/files/000013625.pdf>

# Road to Princess ☆ tree

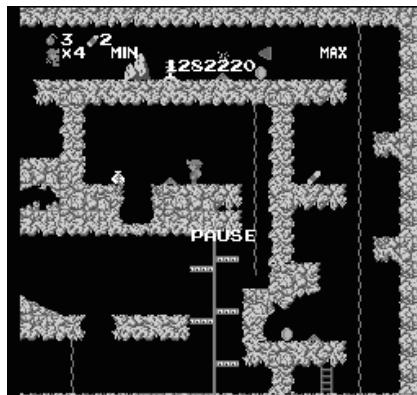
文 編集部 みみずのひもの

## 1 消耗



らく巷の博士前期課程2年生が修論のプランを練り上げに掛かっているであろうこの季節、小生の脆弱な精神は《消耗》によって蝕まれていた。

毎日正午過ぎに起床して研究室に向かいぼんやりと論文を眺め、夜はWORD編集部に立ち寄ってスペランカー(FC版)をおもむろに起動し、コウモリのファンが当たる度に生命を喪失する糞貧弱な探窟家の、袁れで見窄らしい人生を幾度となく憂いた。



累計約5時間弱を不毛に消耗して探窟家を第4階層まで導き、ゲームクリア画面につかの間の喜びを感じたのも今は昔。上昇負荷で人間性を喪失することもないこの探窟家は、2周だろうが3周だろうが無限にダイブできる。だから私は2017年の現代において、攻略サイト<sup>1</sup>を確認しながら今日もファミコンの電源を入れるのである。

安酒をあおり、探窟家を使い捨て、石臼で挽くかのように精神を消耗する日々。

消耗。

一般論として、人はいかにして消耗せずに生きるか？という課題に常日頃から直面している生き物だ。消耗とは人生におけるリソースの無駄遣いであり、基本多くの物事は消耗無く常に最適である事が望ましいと

<sup>1</sup>スペランカー攻略サイトのURLは[こちら](http://greatfulldead.net/supe/index.htm)(<http://greatfulldead.net/supe/index.htm>) 画像もこちらから拝借。なおスペランカーの画像の著作権は全て制作会社のアイレムにあります。

されている。だからこそ人は他人が消耗する姿にはくそ笑み、自分が未だ消耗していない事実に優越感を覚えながら、「まだ〇〇で消耗してるの？」とあざ笑うのだ<sup>2</sup>。

では消耗とは絶対悪だろうか？ そんなはずはない。一般に貪欲法よりも焼きなまし法がよい最適解が得られるのと同じように<sup>3</sup>、一度消耗に陥るからこそ得られる理解もある。重要なのは消耗にある状態から自分が何故消耗しているのか、そしてどう脱するかを常に思索し続けるという心構えだろう。

では如何にして人は消耗を脱するのか？

その解は至極明快であり、結局のところ人が消耗を脱するには新たな消耗を流し込むしか無いのだ。

艦隊が擬人化するやつで消耗していたヲタク達は別の艦隊が擬人化するやつにワッと群がり、紙タバコに消耗していたスマーカー達は IQOS へと切り替え、授業ですら消耗の一種であるとの局地に至った生徒達は日本初の授業をしない塾へと足を運ぶ。そして多くの局面において、それが新たな消耗の始まりであるなどとは微塵も考えない。傍から見れば例え不毛に見えようとも、消耗には消耗をぶつけるのが一番手っ取り早いのだ。

だからこそ結局のところ、意志薄弱な己の消耗に終止符を打つためには、新たな何かで消耗してしまえばいい。何かとびきりに強烈で、刺激的で、何もかもを忘れられる、刹那的快楽による消耗はどこか……。

消耗を渴望する悶々とした頭で、ふと公開されたての雙峰祭のタイムテーブルを眺めていたところ、ツクライブやアイドル研究会に混じって懐かしい名前が目に入った。

その企画は、名を「プリパラ<sup>4</sup>～メモリアルライブ@ツクパ～」と言う。

## 2 回想

あれは 1 年前の雙峰祭、我々はプリンセスツリーなる団体がプリパラのダンスを振りコピする噂を聞きつけて、（中略）見事大興奮の中、その幕を閉じたのである。

## 3 目的

今年もプリパラのコピーユニットであるプリンセスツリーをぶち上げて、全力で消耗していこうという所存である。

## 4 準備

その場に突然赴いても、ヲタクは基本シャイなので何もできない。何事も準備が必要である。今回は以下のように準備した。

<sup>2</sup>ここでの消耗という単語の用法は、例のイケダハヤト氏のブログタイトルの意図するところとは異なり、そこから派生した煽り文句の 1 つであると解釈して頂きたい（予防線）。

<sup>3</sup>ざっくり言うとある最適化問題について、その時点での最高の結果を求めるのが貪欲法、序盤は最適でない方法も色々試し終盤に連れ貪欲法に近づいていくのが焼きなまし法である。

<sup>4</sup>女児向けコンテンツの皮を被った何か。アーケードゲームが全国展開されており、現在も販売促進アニメが放送されている。またタイトルウィスの人体図、麗子像、金剛力士像など多くの美術作品が登場するので、美術教育作品としての完成度も高く評価されている。

## 4.1 スケッチブック

ヲタクとは個の集団であり、一方で演者が求めるのはヲタクの調和された統一性である。演者の意思にそつて個の集団を統率するためには、手段を用いなければならない。その1つがスケッチブックである。

スケッチブックヲタ、通称スケブヲタクはスケッチブックに必要なコールや合いの手など記載し、ライブ中適切なタイミングでそれを掲げる仕事を担う。そうすればライブ中のヲタクは思考が単純なので、それを読み上げるという寸法だ。この時タイミングが遅すぎても早すぎてもぐずぐずになってしまう。そして当然ながら流れる楽曲を知り尽くしたヲタクのみに許され、かつスケブを掲げている間は原則演者側ではなく客席側を向いていなければいけない。小生はこの損な立ち回りを、あえて勝手に引き受けさせて頂くことにした。

使用するスケブはできるだけ遠くまで文字が見えるよう、大きければ大きい方がよい。今回はドンキホーテ秋葉原店でA3サイズスケッチブックを購入した。

## 4.2 サイリウム

電池式ペンライトの輝度が向上した現在、使い捨てサイリウムの使用頻度は減少したかのように思える。

しかし一方でサイリウムには他人に配布しやすいという特性があり、実際アイドル現場の盛り上がりを求める場面でヲタクの調和性を高めるため、配布される事がままある。

今回の公演は夕暮れ時だったため光量が多少強くても問題ないと判断し、ドン・キホーテ秋葉原店で「ルミカライト 大閃光 バイオレット」を2箱（計20本）購入した。

## 4.3 看板

昨年同様、プリンセスツリーのメンバー全員の名前が刻まれている。これは公演の最後に用いる。ダンボール箱を切り貼りし一晩で作成。

## 4.4 曲の予習

コールを間違える失態ほどヲタクにとって恥ずかしい事もない。歌詞とその合間に挟まれる合いの手について、事前に綿密な調査を進めた。

## 5 緊張

@mimizunohimono さんがツイートしました。

「明日のプリンセスツリーライブの事を考えて既に身体が緊張を覚えている。もう深夜やぞ。」

記憶には定かでないが、どうやら前夜からこういった心理状態だったらしい。何故かヲタクは出演者よりも緊張する生き物なのだ。

## 6 異変

そして迎えた11月5日。

直前まで楽曲を繰り返しチェックし、修論中間発表の時よりも緊張した面持ちで、会場のUNITED STAGE（石の広場）まで足を運ぶ。定刻よりも10分ほど延長してダブルダッチの演舞が終わり、最前ヲタクが捌けたので、後部で腕を組んでいた我々が最前中央をどっかりとキープした。この日のために集まったイカレたメン

バーは不肖みみずのひもの、現編集長にして半ヲタ関係者<sup>5</sup>karasu、そして今日のためにわざわざ SHIBUYA から帰郷してきた shinkbr の 3 人だ。また後方では linerlock、らいりゅう号、コンプラ 氏などが見守っている。コンディションは最高の状態まで整ったと自負しても驕りはないだろう。

が、何か様子がおかしい。

まず感じたのは年齢層だ。明らかに筑波大学生よりも若い層が 20 名ほど最前列をキープしている。いや少し若すぎやしないだろうか？ 本来女性の年齢について言及するのは紳士として最も恥すべき行為なので詳細な言及は避けたいが、皆様が明らかに一桁台の年齢である事が素人目にも分かる。

何より最前がつつき女ヲタが 9 割を占めている。実際プリパラ現場は他のアイドルアニメもののライブ会場と比較しても顕著に女性参加率が高く、体感男女比 5:5 ぐらいである事はザラだ。しかしそうだとしても、この圧倒的最前女性がつつき率の高さはどういうことだろうか？

一方で我々最前列よりも後ろにいるヲタクたちは、明らかに年齢層が筑波大学生よりも高かった。プリンセスツリーのパパ面ヲタク<sup>6</sup>だろうか？ いやスマホやビデオカメラを構えている所を見ると、カメコである可能性も拭えない<sup>7</sup>。

「みみずさん、これはひよっとして……」

先程まで編集部でイキ<sup>8</sup>っていた karasuくんも、声を震わせている。ここまでお膳が揃ってやっと小生は、この現場には異変が起きている事に気づく事ができた。

我々は、幼女<sup>9</sup>先輩<sup>10</sup>がプリンセスツリーを応援するちびっこ会場に紛れ込んだ、異形の存在と化していたのである。

## 7 道化

所謂「大きいお友達<sup>11</sup>」が問題視される現象は、今に始まった事ではない。だからこそ大きいお友達の皆は幼女先輩との衝突を避けるため、隠忍自重の精神が根付いたのである。

仮にカースト制度の二等辺三角形を設けるとすれば、最上位に幼女先輩が君臨し、続いて男児先輩、女子中高生先輩、成人女性先輩などが上から鎮座し、我々のようなキモヲタブサメンクソヲタク共は底辺の底の下のアンタッチャブルの部類に属する。故に本来ライブを拝見する行為すら公式からお目零しされていると

<sup>5</sup>ヲタクでありながら出演者の知り合いである者。

<sup>6</sup>普段パパ達は会場の後ろに立ち、腕組みをしながら彼女たちを無言で見守る。

<sup>7</sup>カメラ小僧。現場によっては厄介。アーケードゲームのワンダーモモ（1987 年発表）のおじやまキャラとしてこのカメコが登場している事からも、その歴史の深さがうかがえる。

<sup>8</sup>イキる。意気がる。ライブを前にしてヲタクの気持ちや態度が大きくなること。

<sup>9</sup>女児向けコンテンツにおいて、マーケティングのメインターゲットとされている層。

<sup>10</sup>敬意を払うため先輩という敬称が用いられる。以降当該現場には男児先輩も複数名いらっしゃったが、ここでは若い参加者全体を「幼女先輩」という名称で統一する。

<sup>11</sup>女児向けコンテンツにおいて、マーケティングのサブターゲットとされている層。

いう認識を常に持たねばならないし、プリパラのアーケード筐体に電マを持ち込んだり<sup>12</sup>灰皿が吸いかけのまま放置されるなどの事案<sup>13</sup>は、至極戒めるべき行為としてセンセーショナルに報じられたのである。

プリパラはアニメがどう見ても女児向けではない事から忘れがちだが、あくまでも女児が楽しむコンテンツである。無論小生にとってもそのスタンスを歪めるつもりはない。だからこそ全てを体内から放出して応援するという行為が、公演を楽しんでいる幼女先輩の妨げになるなど以ての外なのだ……。

話を戻そう。1曲目の「Ready Smile !!」が終わった時点で幼女先輩たちから小生に向けられた目線は、驚嘆と好奇に満ちたものだった。

まるで小生は SIREN2 のエンディングで闇人の世界に堕ちた自衛官・永井であり、幼女先輩のお気持ちは異形の存在を目の当たりにした闇人のそれと化していた事だろう。おそらくライブ中に飛んだり跳ねたり叫んだり、跪いて石の床を叩いたり、ステージに向けて咲クラップをする生命体は、今日の今まで既に死滅したものだとは彼女達は思っていたに違いない。



SIREN2 エンディング（永井頼人 24:00）

確か記憶が定かであれば去年開催された神チャレンジライブにおいて、そういった生命体はカンブリア爆発さながらの大増殖を引き起こしていたはずだった。だが彼らはここ1年の間の環境の激変についてこれずに絶滅し、この現場においてはもはや我々 WORD の人間と数名を残すのみとなっていたのだ。

幼女先輩は先程からあくまでも純粋に「何故この人達はこんなにも騒いでいるのだろう？」という目でチラチラとこちらを見ている。ここで karasuくんから提案があった。

「みみずさん、後ろの方に移動しませんか？」

「どういうことだ？」

「そっちでイキったヲタクがジャージャーしてるみたいです」

なるほど。最前からはよく聞き取れないが、きっとステージ後方は地獄絵図と化しているのだろう。そもそも幼女先輩のお気持ちは最優先に考慮するならば、今すぐ視界から消え去るのが人道的判断である。よし緩やかに後退しよう。我々は緩やかに戦略的後退を始めた。

だが我々が後退し、後ろの人々誰も最前のぽっかり空いた空間を埋めようとはしない。それもそうだ。言うなれば我々は農地に劇薬を蒔いてしまった張本人であり、死んだ土地には誰も寄り付かないのは道理だ。かと言って最前列中央に虚無を生成するのも、今度は演者にとってスゴイシツレイに当たるだろう。

<sup>12</sup>アーケードゲーム「プリパラ」で高速振動する何らかの機械（電動マッサージ機）を利用して、不正にスコアを稼いだという事案。公式でアナウンスされた。<http://pripara.jp/news/detail?seq=101>

<sup>13</sup>ソースは Togetter（これしかない）。<https://togetter.com/li/1109613>

もたもたしている間に曲の合間の MC が始まってしまう。

「今日は小さいお友達も、大きいお友達も、来てくれて本当に嬉しいです！」

「大きなお友達」のところでクスクスとした笑い声が漏れる。誰を指しているのかは言うまでもないだろう。

退路は既に絶たれた。

こうなれば、やるしか無い。

プリンセスツリーからも笑われ、幼女先輩からも笑われ、そして会場全体のハコユレ震度を激増させるような存在。小生はスケッチブックを片手に、《道化》になる覚悟を決めたのだ。

## 8 苦戦

ここからは容易に想像できるだろうが、厳しい戦いを強いられたのは言うまでもない。

2曲目「ま～ぶる Make up a-ha-ha!」では息を絶え絶えにしながらアッハッハッハアハハハという高らかな笑い声を腹の奥から吐き出し、4曲目の「でび&えん Reversible-ring」では「そこに跪け」という歌詞に合わせて地に両足で跪き、6曲目の「シュガーレス×フレンド」はジャンケンをテーマにした歌詞なのでジャンケン小僧並のハイテンションでグーとチョキとパーをリズミカルに繰り出した。この辺りは文字起こしをするだけでも険しさが大変際立ってくるので、読者の皆様も実物を目の当たりにしなくて本当によかったと安堵されている頃合いかとお見受けする。無論女児先輩にご迷惑がかからないよう細心の注意を払ったのは言うまでもない。

険しさは留まるることを知らないので、ここで一旦舞台上のプリンセスツリーの面々に話を移そう。

プリンセスツリーの振りコピは実に見事だった。一般にプリパラの振り付けはアニメの3DCGダンスシーンに基づくため、プリパラ世界の女児が軽々と踊れるダンスであっても、現実の人間にとっては振り付けの難しい・激しいものが多い。今年の公演は昨年よりも曲数が増加していたため、やはり心配になるのはスタミナ面だ。しかしプリンセスツリーの皆様はデュオ曲とトリオ曲をうまく織り交ぜることによって6人の疲労度を分散し、それによって昨年よりもダンスのキレを維持できていた様子がうかがえる。

またセットリストもなかなかの良チヨイスだった。数多くのプリパラ楽曲の中でも特にヲタク人気の高い・コールを入れやすい楽曲がおせちのようにずらり並び、の中でも1曲だけアイドルタイムプリパラ<sup>14</sup>の新曲「ブランニュー・ハピネス」を織り交ぜたのは、なかなかのいぶし銀であるとも言えよう。これは前作主人公らあらくんと新作主人公ゆいくんの親睦の深まり具合を示した1曲で、最後のゆめかわ/かしこまという各々の決めポーズシンクロによって、トイパヤボタキたちは次世代プリパラへのバトンタッチ、そして新たな時代の幕開けを否応無しにも痛感してしまうのだ。

30分という短い時間でも十分にぶち上がるプリパラ楽曲の数々、それらの魅力は小さいお子様から大きいお友達まで十二分に伝わった事だろう。

<sup>14</sup>アニメプリパラの続編。前作で神の座まで昇格した旧主人公・真中らあら君と、妄想癖でどこからでも妄想に陥ることができる新主人公・夢川ゆい君のダブル主人公が、パラ宿なる異世界のプリパラを盛り上げたり古代プリパラ滅亡の謎を紐解いたりする新感覚痛快アニメ。

さて。当然ステージが進むに連れ段々と気持ちよくなってくるわけだが、もちろん気持ち良くなるのが自分だけになってはいけない。小生はスケッチブックを観客側に掲げ、時折幼女先輩側にもチラ見せしながら、ここはこういった掛け声をどうかよろしくお願ひします、という懇願の告知を適切なタイミングで施していく。正直レスポンスの掛け声としてはあまり返ってこず、ほぼ自分の声のみが響いていたが、これでいい。幼女先輩のお父様お母様方にとっても、いきなりこいつ何言ってるんだと思われるよりは、自身が何に対してどんなコールをするのか事前に告知した方が断然よい。

そして始めはキヨトンとしていたようだった幼女先輩も、心なしかこのおっさんの奇行に対して少しづつ心を開いてくれている……ような気がした。少なくとも不快感を持って接している感じはしない。緊張と興奮が入り混じり、全身からは冷や汗と熱気が渦巻いたようなぬるい汗が吹き出していた。

## 9 好転

孤軍奮闘が好転したように思えたのは、8曲目「ブランニュー・ハピネス」が終了した頃合いからだろうか。曲が終わってから一旦MCを挟み、その後舞台から捌けたプリンセスツリーの面々は、カラフルなクッションを手にまた壇上へと上ってきた。セットリストは事前に公開されているので、当然次に何が流れるかも知っているわけだが、楽曲の中でクッションを使うシーンが挟まれるとなればもうあれしか無い。次は全体曲「Love friend style」だ。

一般に同じダンスを繰り返し目に焼き付けているヲタクは、当然ながら曲開始時のアイドルの初期配置状態から楽曲名を判別することができる

「ンンッ？ あのフォーメーションはあ～～～♪？」

のような期待に満ちた雄叫びを楽曲開始前に差し込む事がある。

慣例に習って小生も一瞬あのフォーメーションは……と言いかけたが、ヲタクには常にワンパターンを嫌う特性がある。何しろ脳髄はドーパミンの海に溺れて使い物にならないので、脊髄が反射的に声帯を震わせ、「ンンッ？ あのクッションは～～～♪？」

という険しみの極みのような雄叫びが喉から漏れ出た、と反省した時には既に遅かった。

だが完全に想定していなかったが、この限界発言に対して幼女先輩の間からはクスクスという笑いが零れたのだ。男児先輩に至っては「あのクッションは～？」と真似し始める始末だ。

決して不快な人間を嘲笑するためではない、アイドル応援に全力で消耗する滑稽な姿の小生に対して、穏やかな微笑が向けられたのだ。並の現場では到底味わう事の出来ない、自分が笑いものになる事で得られた安心感は、クッションの如く私を安寧で包み込んだ。

私は道化師のフリを演じながら、いつしか心の底まで道化として墮ちる事に悦びを感じていたのである。

## 10 配布

いよいよセットリスト最後の曲「Make it！」が始まろうとしていた。これはプリパラ第一期のOPであり、プリパラで最も最初にステージで流れた始まりの名曲である。

ただちに私はサイリウムの袋を開封した。持参してきたサイリウムは強力な光量を放つものの、持続時間

は僅か 2~3 分と短命だ。最後の曲に最後の盛り上がりを持ち込むには、このタイミングに配布するしかないだろう。

刹那。

幼女先輩にサイリウムを配布する — この時告げられた天啓はここまで流れを考えるとごくごく自然な発想かもしれないが、ある意味では驚きしかなかった。何しろプリパラライブ会場だろうと、i☆Ris 現場であろうと、それを実践に移した人間など見た事がなかったからだ。

もし受け取りを拒否されたら？ という不安が一瞬過る。だがじきに曲は始まる。もはや考えている時間はない。

「みみずさん、やりましょう……！」

karasuくんも小生の行動の意図するところは察したようだ。

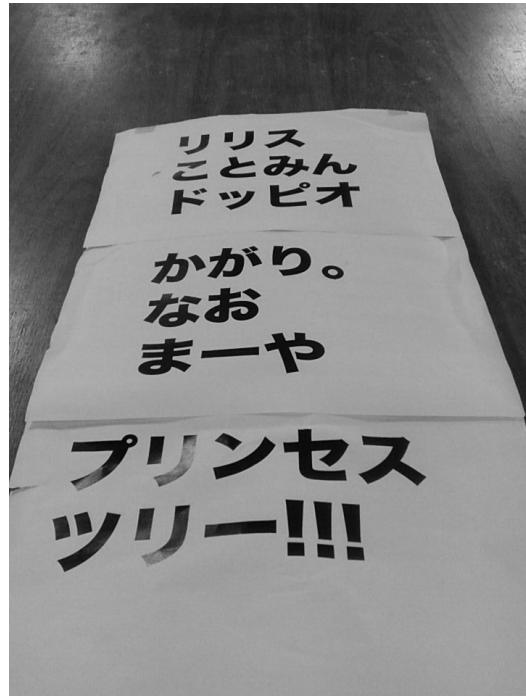
我々は二手に別れ、幼女先輩達にサイリウムの配布を始めた。サイリウムを扱ったことがある諸兄諸姉はご存知かと思われるが、あれは折って発光させるのに意外と力がいる。そこで我々はサイリウムを一度パキり、発光させた状態で手渡すことにした。こうすると幼女先輩も受け取りやすくなる。今後幼女先輩にサイリウムを配布する機会のある各位には、ぜひともこういった点にご注意頂きたい。

思っていた以上にサイリウムは幼女先輩からの人気が高く、イントロは既に始まっていたが、歌い出しまでにはほぼ全ての先輩に配布を完了させることができた。持参したサイリウムでギリギリ数が足りていたのも僥倖の 1 つだろう。

自身のヲタク棒を用意してから改めてステージに向き合った時、そこには男児も女児もパパもママも大学生も、皆一同になってステージを応援する姿があった。アイドルの応援に老若男女の差も、経験も差も表れない世界。みーんなともだち、みーんなアイドル。そこにはまさに「プリパラ」の光景が広がっていたのである。

本物であろうと、コピーユニットであろうと、人を熱狂の渦に巻き込むことができるプリパラはやっぱり凄い。そしてそのライブに、プリパラを象徴するサイリウムという一輪の花を添えられたならば、もはや思い残すことはないだろう。

そしてここまで己を抑制してきた小生ではあるが、気持ちの高ぶりだけは抑えることができない。小生も辛抱堪らず、落ちサビに合わせて背面仰け反り連続咲クラップをステージに向けてかまし、持参の看板を掲げてプリンセスツリーメンバー MIX をブチ上げ、安らかな終焉を迎えたのである。



## 11 終焉

プリンセスツリーのライブは今年も大盛況の内に終わりを迎えた。だがまだ仕事は終わりではない。サイリウムを配布した後には不法投棄を防ぐため、使用済みサイリウムを回収しなければいけないという鉄の掟が存在する。だが幼女先輩から出た要望は、意外にもそれを持ち帰ってもよいかというものだった。

私はハッとなった。普段ヲタク達が消耗品として使い捨てるサイリウムも、子どもたちにとっては1本100円の高級品。ひょっとしたらこれに触れる機会も初めてだったのかもしれない。小生は高校生時代、初めてライブでサイリウムを折った時の事を思わず懐かしみ、どうぞどうぞ持ち帰ってくれと快諾した。どうやら幼女先輩はサイリウムという未知のプレゼントに対して興味津々らしく、最終的に手元にはほとんどのサイリウムが戻らなかった。本来ウリヤヨイヲタク達に消耗されるはずだったサイリウム達が、幼女先輩の笑顔へと姿を変えたのである。ヲタク冥利に尽きるとはこの事だ。



プライバシー保護のためモザイク処

理しています

小生は疲労困憊で顔中から汗を噴出させ、膝から崩れたままの状態でいた。ふと近くに、こちらをじっと見つめるシャイな幼女先輩がいらっしゃる。

先輩ママ「あら～サイリウムもらえたの～よかったですわね～ほら、お兄ちゃんにちゃんとお礼言つて」

先輩「ありがと」

小生「ああ、今日は……楽しかったかい……？」

先輩「うん……楽しかった……」

小生「どの曲が……一番よかったです……？」

先輩「……（無言）……」

小生「そうか……それはよかったです……」

ふとステージ端の向こう側を眺めると、幼女先輩がプリンセスツリーの皆様に群がってやいのやいのしている。後に聞くところによると、ステージ終了後彼女達はお子様から引っ張りだこだったそうだ。

さて、お邪魔虫はひっそり去る時間だ。小生は看板とスケッチブックとサイリウムの容器を手にし、どこともない達成感を胸に抱えたまま会場を去った。

## 12 追伸

この物語はフィクションですが、ちゃんとプリンセスツリーさんは実在します<sup>15</sup>。

プリンセスツリーの皆様、本当にありがとうございました。

<sup>15</sup>幸運にも本誌印刷開始直前に、ステージの様子を録画した動画がアップロードされていました。こちらでライブの様子をご確認ください。<https://www.youtube.com/watch?v=WFS1EcM7Zqg>



図1: プリンセスツリーの皆様と集合写真

# シ ャ ド ウ バ ー ス

## Side1 (2017年)

文 編集部 ド バ

次の OCaml プログラム shadowverse.ml をシャドウバースと呼びます:

```

1. module type X =
2.   sig
3.     module type A
4.     module type F =
5.       functor (_ : sig
6.           module type A = A
7.           module type F = functor(_ : A) -> sig end
8.       end) -> sig end
9.   end ;;
10.  module type Y =
11.    sig
12.      module type A = X
13.      module type F = functor(_ : X) -> sig end
14.    end ;;
15.  module Shadowverse (V : Y) = (V : X)

```

## Side2 (2017年)

シャドウバースをコンパイルします。手元の OCaml コンパイラのバージョンなどを確認<sup>\*1</sup>

```

% ocamlc -version
 4.05.0
% date
2017年 11月 8日 水曜日 16時40分35秒 JST

```

指差し確認ヨシ コンパイルやね

```
% ocamlc shadowverse.ml
(ブーブーン (ファンの回る音))
```

OK Google, シャドウバースのコンパイルが終わったら起こして

---

<sup>\*1</sup> 本記事のものと異なるバージョンの OCaml コンパイラでも試せ（ると思い）ます

### Side3 (2117年)

% date

2117 桶 11 置 r 8i;T海 %GfYN 矢塵 16<限 40@i;ěi;ši;A 35i;Nj;A盈 JST

化けとるね

どうかな？

% ocamlc shadowverse.ml

(繩因?繩ij繩ijij繩ijij繩ijij繩ijij繩ijij繩ijij繩ijij繩ijij繩ijij繩ijij (繩輔い繩i;s鍵i;ö蝗械 k 髮i;s))

は？ お前…………



百年十五行コンパイル於消耗奴…………?

## Side4 (?????????????年)

人間にはやっていきがある 皆さんはやっていきですか？ そこで、シャドウバースのコンパイルが終わらない理由です。

### Pt1: モジュール

コンパイルを終わらせないプログラムでは `module` というキーワードが乱舞していました。えーではね、見ていきたいと思います。説明は大変だが、見ていくことならできるから。

```
module type A = sig
  val x : int
end ;;

module type B = sig
  val x : int
  val y : string
end ;;
```

ここでは モジュール型 A, B を定義しています。モジュール型 A は、`int` 型のフィールド `x`を持ちます。モジュール型 B は、`int` 型のフィールド `x`に加えて、`string` 型のフィールド `y`も持ちます。「モジュール型ってなんやねん……单なる型とは違うの……」というのは立派な質問で、まあ型みたいなもんだと思ってもらつて構いません。C++とか Java でいう「クラス」みたいなものだと思ってもらってもスゴく大きな問題が発生することはないです。というより、クラスを知っているなら「そういう感じ」だと思ってもらった方が、後で `subtyping` つづーもんを説明する時に楽なので、無理のない範疇でそう思ってください。

次は、モジュール型 A を持つ物体（これをモジュールと呼びますが）`ModA` と、同じくモジュール型 B を持つモジュール `ModB` を作ってみます。モジュール型のことをクラスだと思っているそこのあなた、クラス A のインスタンス（オブジェクト）`ModA` を作るみたいなイメージですわ。

```
module ModA : A = struct
  let x = 42
end ;;

module ModB : B = struct
  let x = 10
  let y = "hoge"
end ;;
```

まあこれは構文の話ですので、特に説明できることはありません。OCaml では、ドット演算子を使ってモジュール要素にアクセスできましゅ：

```
# ModA.x;;
- : int = 42
# ModB.x;;
- : int = 10
# ModB.y;;
- : string = "hoge"
```

## Pt2: モジュールを受け取ってモジュールを返す関数（ファンクタ）

関数が大事です。（上で定義した）モジュール型 A から A への関数を定義します。なお、モジュールを引数にしてモジュールを返す関数のことを、OCaml ではファンクタと呼びます。別の言語で例えるならえーと、C++だとテンプレート引数を持つクラスで、Java だと Generic Type (Generics) とかその辺でしょうか。

```
module type AtoA = functor (_ : A) -> A;;
module Twice : AtoA = functor (X : A) -> struct
  let x = 2 * X.x
end;;
```

AtoA はファンクタ型というやつで、A から A へのファンクタの「型」です。このファンクタ型を持つファンクタとして、Twice を定義しとりまして、こいつは受け取ったファンクタのフィールドを二倍にして、新しいモジュールを返します。簡単ですね。よし、じゃあ使ってみます。

```
module ModA84 : A = Twice(ModA) ;;
module ModA20 : A = Twice(ModB) ;;
# ModA84.x;;
- : int = 84
# ModA20.x;;
- : int = 20
```

良いね^^ いやちょっと待ってください。Twice(ModB) を見て。ModB の型は A ではなく B です。それなのに問題なく使えています………… Q. 何故？ A. モジュールにおけるサブタイピングが効いてる

## Side5 (モジュールのサブタイピング年)

モジュール ModA の型はモジュール型 A で、モジュール ModB の型はモジュール型 B で、それぞれ次のように定義していました。

```
module type A = sig
  val x : int
end;;
module type B = sig
  val x : int
  val y : string
end;;
```

なる～～…… これは B による A の **include** 含み じやな<sup>\*2</sup>。なので型 B をもつモジュールを「切り上げ」することにより、いつでもモジュール型 A を作ることができます。OCaml ではこの切り上げを次のようにして確認できます:

```
# module ModAfromB = (ModB : A);;
# ModAfromB.x
- : int = 10
```

一方で、モジュール型 A にはフィールド val y : string がないので、モジュール型 B に変形することはできません。やってみましょう。

```
# module ModBfromA = (ModA : B);;
Error: Signature mismatch:
        Modules do not match: sig val x : int end is not included in B
        The value 'y' is required but not provided
```

はい。

このようなモジュール型 A と B のような関係を指して、B は A の subtype (サブタイプ) であるといい、B <: A と書きます。サブタイプやらサブタイピングの定義は省ますが（そもそも今回は例で使う部分が分かってもらえれば問題ないので）、気持ちとしては

「型 A を持つモジュールが使える箇所ならどこでも、型 B を持つモジュールが使えるならば、B <: A」ぐらいで思ってもらえば大丈夫です。B の方が A よりリッチというか、まあそういうあれ。

OCaml では、二つのモジュール型 X と Y について、Y <: X が成立するかどうかを、次の Shadowverse ファンクタを定義することで確認できます。

```
module Shadowverse (V : Y) = (V : X)
```

これにより、モジュール型 Y を持つモジュール V をモジュール型 X に切り上げられるか？ ということを、OCaml に検査させます。上で見た A と B で、念のためにやってみましょう:

```
module Shadowverse1 (V : B) = (V : A) => 問題なし
module Shadowverse2 (V : A) = (V : B) => 問題あり
Error: Signature mismatch:
        Modules do not match: sig val x : int end is not included in B
        The value 'y' is required but not provided
```

いいですね。

---

<sup>\*2</sup>A の全てのフィールド（今回は x ただ 1 つ）が、B で同じフィールド名と型を持って存在しているとか、そういうニュアンスです。しかもこの場合は A にはないフィールド y が B に入っているので、A が B に真に含まれているとか言いたい。別に言わんでもええけど

## Side6 (ファンクタのサブタイピング年)

ファンクタにもサブタイピングがあります。は～……ではね、、、つっこむぞつかまれッ！

```
module type A = sig
  val x : int
end ;;

module type B = sig
  val x : int
  val y : string
end ;;

module type F_sig = functor (_ : A) -> A ;;
module type G_sig = functor (_ : B) -> A ;;
module F : F_sig = functor (X : A) -> struct
  let x = X.x
end ;;

module G : G_sig = functor (X : B) -> struct
  let x = X.x + String.length X.y
end ;;
```

ドバーッとな まあ細かい説明はいらんでしょ。ファンクタ型 `F_sig` ( $= A \rightarrow A$ ) と ファンクタ型 `G_sig` ( $= B \rightarrow A$ ) では、`F_sig <: G_sig` と `G_sig <: F_sig` のどちらかが成立します。どちらでどうか？ 既にみた通り  $A <: B$  なので、`G_sig <: F_sig` でしょうか？

答えは次の通り：

```
module Shadowverse1 (V : F_sig) = (V : G_sig) => 何も言わないので F_sig <: G_sig
module Shadowverse2 (V : G_sig) = (V : F_sig) => 問題あり
Error: Signature mismatch:
  Modules do not match:
    B -> sig val x : int end
    is not included in
      F_sig
    At position functor (_ : <here>) -> ...
  Modules do not match: A is not included in B
    At position functor (_ : <here>) -> ...
  The value 'y' is required but not provided
```

こらどういうことでしょうか。なんで `F_sig <: G_sig`？ `F` と `G` を実際に使って、ちょっと説明させてね：

```

module MA : A = struct let x = 20 end (* 型を A にするモジュールの定義 *)
module MB : B = struct let x = 20      (* 型を B にするモジュールの定義 *)
                     let y = "hoge" end

module MA1 : A = F(MA)  (* OK *)
module MA2 : A = F(MB)  (* OK *)
module MA3 : A = G(MB)  (* OK *)
module MA4 : A = G(MA) => 問題あり

Error: Signature mismatch:
        Modules do not match: sig val x : int end is not included in B
        The value 'y' is required but not provided

```

`G(MA)` でエラーが起こるのは当然で、というのは `G` が引数でモジュール型 `B` を要求するのに対して、`MA` の型は `A` だからです。`MA` にはフィールド `y` がないので、`G` の計算ができません。

`G(MA)` で型に関するエラーが起こる一方で、`F` はどちらの引数に対しても成功します。言い換えると

- `G_sig` 型のファンクタ `G` が適用できる場所 `G(X)` では、`F_sig` 型のファンクタ `F` を適用できる `F(X)`
- ただし、その逆は成立しない (`F(MA)` は良くて `G(MA)` はダメ)

モジュール (型) の時にも「型を `A` とするモジュールが使えるところではどこでも、型を `B` とするモジュールが必ず使えるならば `B <: A`」と書きましたが、ファンクタ (型) におけるサブタイピングも同様です。従つて `F_sig <: G_sig` となります。

こことこは大事なので、見栄え良くまとめておきますが

$$\frac{B <: A}{\text{functor } A \rightarrow C <: \text{functor } B \rightarrow C} \quad (\star)$$

上が成立するなら下が成立すると読みます。`B <: A` ならば `functor A -> C <: functor B -> C` が成立、です。

《この括弧の中は、知っている人向けの確認なので、分からぬ人は一切気にしないでください。

はいそうですね。一般的には

$$\frac{B <: A \quad C <: D}{\text{functor } A \rightarrow C <: \text{functor } B \rightarrow D}$$

こういう事になつります。》

## Side7

元のプログラムが停止しなかったことを確認する、その思いを胸に次の例をみてみます。

```

module type Int = sig
  val x : int
end ;;

```

```

module type String = sig
  val x : string
end ;;

module type X = sig
  module type A
    moduleToStr : functor (_ : A) -> String
  end ;;

module type IntX = sig
  module type A = Int
    moduleToStr : functor (_ : A) -> String
  end ;;

module Shadowverse (V : IntX) = (V : X) (* OK *)

```

`Int` と `String` はそれぞれ `int` 型と `string` 型のフィールド `x` を持つモジュール型です。

モジュール型 `X` は、「正体不明の」モジュール型 `A` と、`A` からモジュール型 `String` へのファンクタ `ToStr` からなります。モジュール `IntX` は、`X` を特殊化したもので、`A` には `Int` が割当てられていると考えられます。この時は `IntX <: X` が成立するのですが、どのような計算に基づくのか、それを見ていきます。今回のようにモジュールの中に複数の項目がある場合には、フィールドを上から順に処理します：

1. `IntX <:1 X` (`<:1` はすぐ後で見ますが、`module type A` に関する判定) が成立し
2. `IntX <:2 X` (`<:2` はすぐ後で見ますが、`functor ToStr` に関する判定) が成立するなら
3. `IntX <: X` が成立する。

それぞれの計算内容は以下の通りです：

```

IntX <:1 X
⇒ module type A = Int <: module type A
⇒ 左側の方がより具体的なモジュール型なので成功

```

成功したので計算を続けます。モジュール `X` 側の `A` に `Int` を「代入して」残りの部分の計算を行います。すなわち：

```

IntX <:2 X
⇒ (moduleToStr : functor (_ : Int) -> String)
     IntX 中では A = Int が分かっている
<: (moduleToStr : functor (_ : Int) -> String)
     X 中では A の正体は不明だが、ここでは
     IntX 中の A = Int の情報で具体化している
⇒ 両方のファンクタの型が同じになっているので成功

```

両方の結果が成功となったので、`IntX <: X` 自体も成功ということになります。

## ここまでの一例を踏まえて、シャドウバース

シャドウバースを思い出してください（意味を変えずに、わかりやすさのために少しだけ書き換えています）

```
module type E = sig end;;
module type X = sig
  module type A
  module type F = functor (_ : sig
    module type A = A
    module type F = functor(_ : A) -> E
  end) -> E
end;;
module type Y = sig
  module type A = X
  module type F = functor (_ : A) -> E
end;;
module Shadowverse (V : Y) = (V : X)
```

module Shadowverse (V : Y) = (V : X)、すなわち  $Y \triangleleft X$  の検査は、これらのモジュール型の構成要素が二つであるため、 $Y \triangleleft_1 X$  と  $Y \triangleleft_2 X$  に分けられます（参照: Side7）。

まず  $Y \triangleleft_1 X$  ですが

```
Y <:1 X
⇒ module type A = X <:1 module type A ⇒ この形は成功
```

ということで成功します。

$Y \triangleleft_2 X$  は、 $A = X$  を大事にしつつ、以下のように展開します：

```
Y <:2 X
⇒ module type F = functor X -> E
<: module type F = functor (sig module type A = X;
  module type F = functor A -> E; end) -> E
  Y 中の A=X の情報で具体化
```

ファンクタ型の検査についてなので、Side6 の規則（★）により、次のステップに進みます

```
(sig module type A = X; module type F = functor A -> E; end) <: X
```

$Y$  と  $(\text{sig module type A = X; module type F = functor A -> E; end})$  は同一のものですから、もとの  $Y \triangleleft X$  の検査に戻ってきた事になります。すなわち

$Y \triangleleft X \Rightarrow \dots \Rightarrow Y \triangleleft_2 X \Rightarrow \dots \Rightarrow Y \triangleleft X \Rightarrow \dots \Rightarrow Y \triangleleft_2 X \Rightarrow \dots \Rightarrow Y \triangleleft X \Rightarrow \dots$

こんなループがコンパイルの裏側で起こってたんじゃあ、そら 100 年経っても止まらんわね。

## おわりに

以上、OCaml ではコンパイルが終わらないこともあるという話でした。コンパイルが終わらない話自体は別に目新しいものではなく、それこそ今回の話に限れば明確なソースがあります。まず Andreas Rossberg (モジュールシステムの研究をしている方) が今回使ったものと同じ例を OCaml のメーリングリストに 1999 年に流しました。これがそれです:

**Undecidability of OCaml type checking**, Andreas Rossberg, 1999

[https://caml.inria.fr/pub/old\\_caml\\_site/caml-list/1507.html](https://caml.inria.fr/pub/old_caml_site/caml-list/1507.html)

この時から今日まで、OCaml のコンパイラはこの例でループし続けます。従って、OCaml のバグというわけではないです。

本記事で扱ったプログラムの元ネタが、Rossberg の投稿したプログラム……とも言えるのですが、元ネタの元ネタがあります。次の論文です:

**A type-theoretic approach to higher-order modules with sharing**

Robert Harper & Mark Lillibridge, POPL 1994

Higher-order modules with sharing というモジュールシステムに関する話です。この論文で、今回使った例が型検査を難しくするような原因であるとして紹介されており、さらにこの論文では、型検査が決定不能（与えられたプログラムが与えられた型を持つかどうかを判定するアルゴリズムが存在しない）という結果まで証明しています。言い方を変えると、型検査（コンパイル）中にチューリング機械を動かして、色々な計算をさせることができます。ただこの論文で使われているチューリング機械の模倣手段が、今日の OCaml でも使えるかどうか、これは OCaml のモジュールシステムの全体像を把握していないためそこまでは分かりません。（多分できると思いますが）

型検査が決定不能というのは、なんというかまあ悪いイメージがあるかもしれません、これは逆に言えば OCaml のモジュールシステムは強力で、色々なプログラムを表現できるということになります。そもそも、今回みた結果のあるなしに関わらず我々はガンガン OCaml を使ってきましたし、これからもガンガン OCaml を使っていきましょう。ちなみに、Java の型検査も決定不能です:

**Java generics are Turing complete**, Radu Grigore, POPL 2017.

情報科学類誌

WORD

From College of Information Science

農水省も「一太郎」より  
WORD!号

発行者

情報科学類長

編集長

吉本 祐樹

筑波大学情報学群

情報科学類 WORD 編集部  
(第三エリア C 棟 212 号室)

制作・編集

2018年3月31日 初版第1刷発行 (256部)