TSNET スクリプト通信

TSC 編集委員会

ISSN 1884-2798

目次

巻頭言	jscripter ...	3
GAawk	Y さ ...	4
追録の加除	ムムリク ...	24
よしおさんとロボ太　全編	海鳥 ...	30
スクリプトでパズルを可視化する	海鳥 ...	73
覚醒と進化 試論 I	jscripter ...	91
編集後記	jscripter ...	97

表紙写真:　謎の船
撮影:　　jscripter
日時:　　2011 年 10 月 9 日
場所:　　宇品 10 万トンバースにて
メモ:　　停泊している海上保安庁巡視船「ぎんが」の向こうに奇妙な謎の船が現れた。マツダに向かった模様。

巻頭言

jscripter

さて、編集者の原稿で遅れた第 14 号。ようやく完成の運びに。編集者は風邪もあってお疲れ気味だが、内容は豊富になった。

Y ささんの「GAawk」の GA とは遺伝的アルゴリズムの意。TSNET スクリプト通信ではありとあらゆることが勉強できるのである。

ムムリクさんの「追録の加除」。加除式本の存在は知ってはいるが、自分で管理したことはない。Ruby on Rails は世界を救うためにあるのだ。

「よしおさんとロボ太」は今回、掲載終了記念として全編一挙収録して、みなさんにまとめて読んでいただけるようにしてみた。楽しんでください。

海鳥さんからは、数独のパズルを富豪的プログラミングで解く記事を投稿していただいた。「 Ruby と Graphviz で数独」が副題である。

さて、編集者は、よたよたとしていて、結局大上段に構えたのはよいが、振り下ろす先をここ二日間探りつつ、スクリプトはないけど、なんとか方向を見出した。これはほとんど思索と偶然のなせる技である。XTM や DITA で TSNET スクリプト通信や更新日記が書けるようになるとおもしろいかもと考えている。

以上。

(投稿: 2011 年 11 月 20 日)

GAawk

written by Yさ

1. この記事について

遺伝的アルゴリズム(Genetic Algorithms, GA)ってご存じですか？
簡単に言うと、進化によって問題の答えを解く、というものです。
本稿はGAをawkでやってみた事について、書いて残しておきたいと思います。
...gawk4.0.0を使ってみたかっただけなのですが(;^^ゞ

※タイトルが某『飛行機をロボットに!?』でお馴染みの可動膝関節による有翼地面効果支援兵器にとても良く似た感じになってますが、全く関係ありません。そちらは GERWALK でスペルが違います。

2. 参考文献

本稿はWeb上で検索して見つかった色々なものを参考に記述しています。中でも特に参考としたURLを以下に挙げます。

1) 遺伝的アルゴリズム
http://www.obitko.com/tutorials/genetic-algorithms/japanese/
(注: 日本語が一部怪しげ...)

2) マッチ箱の脳 (WEB) 篇 / マッチ箱で作るGA#1～#3
http://www.1101.com/morikawa/index_AI.html
(↑ ほぼ日刊イトイ新聞 http://www.1101.com/)

3. GAとは？

ダーウィンの進化論をモチーフとして、
・色々な個体がある
・環境に応じて、より優秀な個体だけが子孫を残し、劣等な個体は淘汰される
・次世代に残った個体は突然変異を起こす場合がある
・こうしたことを繰り返して、進化していく
この流れで優秀な個体＝(良い)解答を見つけ出します。

1) まず始めに、解きたい問題を「遺伝子」で表現します。
例えばbit列(バイナリ)や文字列などです。

この遺伝子を持つものを生物のような1つの個体と考え、いろいろな遺伝子を持つ固体を作ります。

2) 次にそれぞれの個体が持つ遺伝子がどのくらい優秀かを調べます。
その優秀さに応じて順位をつけます。順位が上の遺伝子を残し、そうでない遺伝子は淘汰します。

3) 生き残った個体は、お互いの遺伝子を交わらせて子孫を残します。
劣等な遺伝子がいなくなった代わりに子孫が増えて、新たな世代となります。

4) 再び、各遺伝子の優秀さを評価し、優秀な遺伝子は生き残り、ダメな遺伝子は淘汰される. . .

を繰り返します。こうして世代をどんどん入れ替えていくと、最後には「最も良い」解答にたどり着けます。
というのがGAの大まかな仕組みです。

■GA用語集

・遺伝子(Gene) :
個体の形質を規定する基本構成要素
・染色体(Chromosome) :
複数の遺伝子の集まり
・染色体長(Chromosome length) :
染色体の長さ＝遺伝子の個数
・個体(Individual) :
染色体によって特徴づけられた個 (※シンプルなものは1固体＝1染色体です)
・母集団(Population) :
個体の集まり
・集団サイズ(Popolation Size) :
集団内の個体数

・遺伝子型(Genotype) :
染色体の内部表現 (文字列, 木構造 or グラフ)
・表現型(Phenotype) :
染色体によって規定される形質の外部的表現 (※解こうとする問題)
・コード化(Encoding) :
表現型から遺伝子型へ変換すること
・デコード化(Decoding) :
遺伝子型から表現型へ変換すること

・適応度(Fitness) :
環境に対する個体の適応の度合 (基本的には"表現型"で評価する)
・遺伝的操作(Genetic Operation) :
交叉、突然変異などの操作 (基本的には"遺伝子型"で操作する)
・選択(Selection) :

適応度に基づく個体の選択淘汰
・交叉(Crossover) :
個体間での遺伝子の入れ換え
・交叉率(Crossover rate) :
交叉に参加する個体が母集団のうち何割かを定める（※異なる定義あり）
・突然変異(Mutation) :
遺伝子の確率的な置き換え
・突然変異率(Mutation rate) :
各個体について突然変異が起こる確率（※異なる定義あり）
・世代(Generation) :
遺伝的操作の１サイクル、または、ある時点での集団を構成する個体群
---

4. GA のロジック

ロジック的には、一例として次のような流れとなります。
---
1. [Initialization] 初期集団の生成
2. [Evaluation] 集団の評価
  2.1 [Fitness] 適応度の算出
  2.2 [Termination] 終了判定
3. [Reproduction] 繁殖（次世代の集団の生成）
  3.1 [GeneticOperation] 遺伝的操作
    3.1.1 [Selection] 選択
    3.1.2 [Crossover] 交叉
    3.1.3 [Mutation] 突然変異
4. [Loop] 2 へ戻る

・[Initialization]:
問題に応じた「遺伝子」を決定した後、母集団の初期化を行います。
本物の生態系でも多様性が大切だといわれているのと同じ理由で、GA においても、最初の世代には、いろいろなタイプの遺伝子を持つ固体がいる必要があります。
一般的には持つ遺伝子をランダムに決めます。

・[Fitness] :
各個体(＝遺伝子)の「成績」＝適応度を算出します。どう算出するかは問題ごとに異なります。
成績によって生き残る個体、淘汰される個体が選ばれます。個体の中で問題(環境)への適合度の高い個体は増殖し、逆に適合度の低い個体は淘汰されます。

・[Selection]：
生き残った個体のうちからは、次の世代に子孫を残せる親となる資格を持つ個体が選ばれます。親となる個体が決定したら彼らの子供を作ります。
ただし、これが完全に親の生き写し＝コピーだけだと進化とならないので、交叉(Crossover)という方法で両親の遺伝子を交ぜ合わせたり、突然変異(Mutation)を起こして遺伝子の一部をランダムに変化させたり、といった操作をします。

・[Crossover]：
遺伝子を交ぜ合わせること、つまり2つの染色体をある部分で入れ替えることを意味します。
どの個所で交叉するかは、一般的にはランダムに決めます。
また、交叉するポイント数も、何カ所でも構わないことになっています。

・[Mutation]：
ある個所の遺伝子が変化してしまうことをいいます。
突然変異を起こすかどうか、どの個所で突然変異するかは、これもまたランダムに決めます。

・[Termination]：
GAは先に説明したように、
①各個体の成績を出す
②成績によって、親となる個体、淘汰される個体を選ぶ
③親となった遺伝子を交叉、突然変異させて、新しい個体(子)を作る
を、繰り返します。これを、一世代と呼びます。
繰り返しを終了させる一般的な条件には、
a. 最上位/最下位の固体が目標とする適応度に達した
b. 「世代」発生を設定した上限まで繰り返した
c. もほや進化が望めなくなった
（例えば次世代の適応度が一定期間続けて向上しなくなった）
といったものがあります。

■GAの主要パラメータについて

・集団サイズ(Popolation Size)：
この数は個体群(１つの世代)の中にどれだけの染色体があるのか、を意味します。
が、大きな個体群にしたとしてもGAのパフォーマンス(解を見つける速さという目的で)を改善させることは無いそうです。
良い個体群の大きさは20～30とのことです。50～100が最も良いというケースもあるそうです。また、ある研究によると最も良い個体群の大きさはコード化すなわちコード化された文字列の大きさに依存するとされています。

・交叉率(Crossover rate)：

交差率は一般的にほぼ80%～95%のように高く設定します。
(しかし問題によっては交差率は60%が最も良い、とのことです)

・突然変異率(Mutation rate)：
その一方、突然変異率は非常に低くします。
最も良い率は、だいたい0.5%～1%とのことです。

5. GAのサンプル

GAを使っていくつかの問題を解いてみます。(もっともGAを使う必要が全くない問題ですが...)

○その1：

比較的エンコード/デコードがシンプルな、表現型と遺伝子型が近いパターンです。

問題) 回文数とは、101や12321といった逆から数字を読んでも同じ数になる数のことである。 5を掛けると回文数になる3桁の最大の整数を求めよ。

→以下の考え方で問題を解いてみます。

・遺伝子/染色体/個体
100～999の3桁の整数を染色体とします。

・個体数/初期集団
20個の3桁のランダムな整数とします。

・適応度
以下で算出します。
1) 5を掛けると回文数になる場合、 +1000 +それ自身の数
それ以外は、0

・選択
0より大きい適応度の上位20%(4個)はそのまま次世代に残すとします。
残りは両親を全体からランダムに選択し、交叉で増やします。

・交叉
交叉率は既に両親の選択時に絞り込んでいるので100%とします。
交叉場所を以下の3パターンでランダムに決めます。
a. 2桁目以降 [-**]
b. 3桁目のみ [--*]
c. 2桁目のみ [-*-]

親のお互いの'*'の数字を入れ替えて、子を生成します。

・突然変異
突然変異率は高めの 10%とします。全体からランダムに個体を選択します。
選択した個体のいずれかの桁をランダムに 0～9 に変化させます。
ただし先頭桁は 1～9 とします。

・終了条件
100 世代に達するまで繰り返す、とします。

→実際に実行してみた結果は、大抵の場合は"正解"を見つけることができました v(^^;
ランダムなアルゴリズムなので見つけられない場合もある、という事です...

○その 2 ：
エンコード/デコードがやや複雑な、表現型と遺伝子型が離れているパターンです。

問題) 覆面算とは、数字(0～9)を文字に置き換えた計算式のパズルである。
ルールは次の 2 つ：
・各文字には 0 ～ 9 の数字が入り、同じ文字は同じ数字となる。
・各項の先頭桁には 0 が入ることはない。

次の式に数字を当てはめよ。
① SEND + MORE = MONEY ［文字種 8］
② WOODS + WOODS + WOODS = FOREST ［文字種 8］
③ EARTH + AIR + FIRE + WATER = NATURE ［文字種 10］

→以下の考え方で問題を解いてみます。

・遺伝子/染色体/個体
重複しない 0～9 から成る長さ 10 の文字列を染色体とします。
染色体の先頭から順に式に使用している文字に当てはめます。
問題(使用している文字数)によっては未使用の数字があります。

・個体数/初期集団
30 個とします。10 個は次の順列をシフトしたもの、とします。
0,1,...,8,9
→ 9,0,...,7,8
→ 8,9,...,1,2
...
→ 1,2,...,9,0

残りの 20 個は 0～9 を 1 回づつ使用したランダムな数列とします。

・適応度
以下の値の和とします。
1) 計算が成り立つ場合、+1000
2) 各項の先頭桁>0 か？ +(0 以外であるべき先頭桁数 -先頭桁が 0 の数) *100
3) 計算誤差 +0～99

・選択
適応度の上位 20%(6 個)はそのまま次世代に残すとします。
また 20%(6 個)はランダムな数列とします。
残りは交叉によって次世代を生成します。
交叉のペアを、一方を適応度の上位 30%(9 個)から 1 つランダムに選択し、
もう一方を全体からランダムに選択します。

・交叉
交叉率は 80%とします。(つまり 20%は元のままのコピー、となります。)
いわゆる"順序交叉"を行うとします。
1) 親 A をランダムに 1 点で切り、子 1 に親 A の切れ目より前の遺伝子をコピーする。
親 A の残りの遺伝子を、親 B に現れる順番にコピーする。

親 A[2, 4, | 1, 3, 5] → 子 1[2, 4, | 3, 5, 1]
親 B[3, 2, | 5, 4, 1] → 子 2[3, 2, | 4, 1, 5]

2) 子 2 は、親 A と親 B を入れ替えた操作を行い生成する。

・突然変異
突然変異率は 10%とします。
個体のいずれかの 2 桁をランダムに入れ替えます。
ただし適応度の上位 2 個は変異の対象外としてます。

・終了条件
300 世代繰り返したら、終了とします。

→実際に実行してみた結果は、たまに"正解"を見つけることができるといった感じで、ほとんどの場合は見つけることができませんでした(;^^ゞ
交叉ロジックを見直すとか、各パラメータを調整すると改善されるかも...

6. スクリプトとその動かし方

・GA.awk --- フレームワーク（他のスクリプトで"@include"して使います）

```
##  GA.awk（フレームワーク） written by Ｙさ

function rnd(N){ return int(N * rand()); }  ##  乱数

# 主要変数
# P      # 個体数
# Cr     # 交叉率(%)
# Mr     # 突然変異率(%)
# G      # 世代
# DNA[]  # 遺伝子/染色体/個体
# Eval[] # 適応度
# Idx[]  # 順位

# ↓以下の各 function を本体に実装してください。
# rz_Initialization="Initialization"; # 初期集団の生成
# rz_Result="Result";                 #   結果の表示
# rz_Fitness="Fitness";               # 適応度の算出
# rz_Termination="Termination";       # 終了判定
# rz_Selection="Selection";           # 選択
# rz_Crossover="Crossover";           #   交叉
# rz_Mutation="Mutation";             # 突然変異
# rz_Regeneration="Regeneration";     # 再生（※次世代を現世代の配列にコピー）

BEGIN{
  # ↓以下は本スクリプトにありますが、必要に応じて置き換えてください。
  rz_Evaluation="Evaluation";         # 集団の評価
  rz_Sort="sort";                     #   順位付け
  rz_Reproduction="Reproduction";     # 繁殖（次世代の集団の生成）

  srand();
}

# ↓以下は必要に応じて、変数 rz_Setup, rz_Finish に設定して実装してください。
# function setup(){}   # 事前準備
# function finish(){}  # 後片付け



#
# メインループ
```

```
# ↓以下を本体から呼びます
function start(){
  if(rz_Setup!="") @rz_Setup();

  @rz_Initialization();
  for(G=1; @rz_Evaluation()==0; ++G)
    @rz_Reproduction();

  exit;
}
END{
  @rz_Result();
  if(rz_Finish!="") @rz_Finish();
}



#
# 集団の評価
#
function Evaluation(  t){
  for(t=1; t<=P; ++t) Eval[t]=@rz_Fitness(DNA[t]);
  @rz_Sort();
  if(G==1) @rz_Result();
  return @rz_Termination();
}
# 順位付け
function sort(  n){
  for(n=1; n<=P; ++n) Idx[n]=n;
  asort(Idx, Idx, "cmp");
}
function cmp(i1,v1,i2,v2){
  if(Eval[v1]>Eval[v2]) return -1;
  if(Eval[v1]<Eval[v2]) return 1;
  return 0;
}



#
# 繁殖（次世代の集団の生成）
#
function Reproduction(  t,nxDNA){
  delete nxDNA;
```

```
  @rz_Selection(nxDNA);
  for(t=1; t<=P; ++t){
    @rz_Mutation(t,nxDNA);
    @rz_Regeneration(DNA,nxDNA,t);
  }
}
```

・palindromic.awk --- 回文数サンプル

1)<回文数サンプル>
>gawk -f palindromic.awk[Enter]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

等として、実行してください。
初期集団と終了時の集団の適応度などの表示を行います。
おまけとして、finish()で総当たりを行って見つかった"正解"を表示します。

```
@include "GA.awk"


##  palindromic.awk (回文数サンプル)  written by Ｙさ

BEGIN{
  rz_Setup="setup";
  rz_Finish="finish";

  rz_Initialization="Initialization"; # 初期集団の生成
  rz_Result="Result";                 #   結果の表示
  rz_Fitness="Fitness";               # 適応度の算出
  rz_Termination="Termination";       # 終了判定
  rz_Selection="Selection";           # 選択
  rz_Crossover="Crossover";           #   交叉
  rz_Mutation="Mutation";             # 突然変異
  rz_Regeneration="Regeneration";     # 再生 (※次世代を現世代の配列にコピー)

  start();
}

# 事前準備
function setup(){
  P=20;    # 個体数
  Cr=100;  # 交叉率(%)
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
  Mr= 10;  # 突然変異率(%)
}

# 後片付け
function finish(){
  # (おまけ)
  for(t=999; t>=100 && !isPalindromic(t*5); --t)
    ;
  printf("¥n< %d *5 = %d >¥n", t, t*5);
}



#
# 初期集団の生成
#
function Initialization(  t){
  for(t=1; t<=P; ++t) DNA[t]=rnd(900)+100;
}

# 結果の表示
function Result(  t){
  printf("¥n[%d]¥n", G);
  for(t=1; t<=P; ++t)
    printf("%2d: %d *5 = %4d (%d)¥n",
      t, DNA[Idx[t]], DNA[Idx[t]]*5, Eval[Idx[t]]);
}

#
# 適応度の算出
#
function Fitness(x){
  if(isPalindromic(x*5)) return 1000+x;
  return 0;
}
function isPalindromic(x,  n,s,p){
  s=sprintf("%d", x);
  n=length(s);
  for(p=1; p<n+1-p; ++p)
    if(substr(s,p,1)!=substr(s,n+1-p,1)) return 0;
  return 1;
}
```

```
#
# 終了判定
#
function Termination(){ return (G>=100); }


# 再生 (※次世代を現世代の配列にコピー)
function Regeneration(dst,src,t){
  dst[t]=src[t];
}



#
# 選択
#
function Selection(nxDNA,  t,n,n1,n2){
  for(t=1; t<=int(P*0.2); ++t){
    if(Eval[Idx[t]]>0) nxDNA[++n]=DNA[Idx[t]];
  }
  if(n>0 && ((P-n)%2)) nxDNA[++n]=nxDNA[1];
  for(t=n+1; t<P; t+=2){
    n1=rnd(P)+1;
    do{ n2=rnd(P)+1; }while(n1==n2);
    @rz_Crossover(n1, n2, DNA, t, t+1, nxDNA);
  }
}

#
# 交叉
#
function Crossover(p1,p2,src,c1,c2,dst,  r){
  if(rnd(100)>=Cr) return;

  r=rnd(3);
  # a. 2桁目以降 [-**]
  if(r==0){
    dst[c1]=int(src[p1]/100)*100 + src[p2]%100;
    dst[c2]=int(src[p2]/100)*100 + src[p1]%100;
  }
  # b. 3桁目のみ [--*]
  if(r==1){
    dst[c1]=int(src[p1]/10)*10 + src[p2]%10;
```

```
    dst[c2]=int(src[p2]/10)*10 + src[p1]%10;
  }
  # c. 2桁目のみ [-*-]
  if(r==2){
    dst[c1]=int(src[p1]/100)*100 + (int(src[p2]/10)%10)*10 + src[p1]%10;
    dst[c2]=int(src[p2]/100)*100 + (int(src[p1]/10)%10)*10 + src[p2]%10;
  }
}

#
# 突然変異
#
function Mutation(t,src,  r,x){
  if(rnd(100)>=Mr) return;

  r=rnd(4);  x=src[t];
  # 1桁目が変化
  if(r==0){ src[t]=(rnd(9)+1)*100 + x%100; }
  # 2桁目が変化
  if(r==1){ src[t]=int(x/100)*100 + rnd(10)*10 + x%10; }
  # 3桁目が変化
  if(r==2){ src[t]=int(x/10)*10 + rnd(10); }
  # 全桁変化
  if(r==3){ src[t]=rnd(900)+100; }
}
```

・alphametic.awk --- 覆面算サンプル

2)<覆面算サンプル>

```
>gawk -f alphametic.awk[Enter]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

等として、実行してください。

問題「SEND + MORE = MONEY」について、初期集団と終了時の集団の適応度などの表示を行います。

また、おまけとして、BEGIN部の「rz_Finish="finish"」を設定している箇所のコメントを外しておくと、finish()で総当たりを行って見つかった"正解"を表示します。

他の問題を解く場合はsetup()で変数FORMULAを設定している箇所を書き換えるか、

```
-v FORMULA="KYOTO + OSAKA = TOKYO"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

といった感じでオプションを付けて起動してください。

```
@include "GA.awk"


##  alphametic.awk (覆面算サンプル)  written by Ｙさ

BEGIN{
  rz_Setup="setup";
  # rz_Finish="finish";

  rz_Initialization="Initialization"; # 初期集団の生成
  rz_Result="Result";                 #   結果の表示
  rz_Fitness="Fitness";               # 適応度の算出
  rz_Termination="Termination";       # 終了判定
  rz_Selection="Selection";           # 選択
  rz_Crossover="Crossover";           #   交叉
  rz_Mutation="Mutation";             # 突然変異
  rz_Regeneration="Regeneration";     # 再生 (※次世代を現世代の配列にコピー)

  start();
}

# 事前準備
function setup(){
  P=30;   # 個体数
  Cr=80;  # 交叉率(%)
  Mr=10;  # 突然変異率(%)

  # 問題の分解
  #   WORD[], WordCnt, Top[], TopCnt,  Use[], Cnv[], GeneSize
  if(FORMULA=="") FORMULA="SEND + MORE = MONEY";
  WordCnt=split(FORMULA, WORD, /[ ¥+=]*/);

  delete ch;
  for(n=1; n<=WordCnt; ++n){
    sz=length(WORD[n]);
    for(p=1; p<=sz; ++p) ++ch[substr(WORD[n],p,1)];
  }
  delete Use;
  delete Cnv;
  for(t in ch) Use[GeneSize++]=t;           # Use[遺伝子の位置] = 使用文字
  if(GeneSize>10){
    print "impossible condition  [", GeneSize, "]";
```

```
    exit;
  }
  for(n=0; n<GeneSize; ++n) Cnv[Use[n]]=n;  # Cnv[使用文字] = 遺伝子の位置
  delete ch;
  for(n=1; n<=WordCnt; ++n) ++ch[substr(WORD[n],1,1)];
  delete Top;
  for(t in ch) Top[TopCnt++]=t;
}

# 後片付け
function finish(){
  # (おまけ)
  delete SAVE;
  for(n=0; n<10; ++n) num[n]=n;
  sequence(num,9);
}
function sequence(num,n,  p,x){
  if(n<0){
    if(wrongZero(num)==0 && calcTerm(num)==calcAns(num)){
      if(NotYetDisplay(num)){
        print "";
        for(x=0; x<GeneSize; ++x)
          printf("%s:%d ", Use[x], num[x]);
        display(num);
      }
    }
  }else{
    for(p=n; p>=0; --p){
      swap(num, n, p);
      sequence(num, n-1);
      swap(num, n, p);
    }
  }
}
function NotYetDisplay(num,  key,n){
  for(n=0; n<GeneSize; ++n) key=key num[n] "";
  if(! (key in SAVE)) return (SAVE[key]=1);
  return 0;
}


#
```

```
# 初期集団の生成
#
function Initialization(  t,n,p,x){
  for(n=0; n<10; ++n) DNA[1][n]=n;
  x=1;
  for(t=2; t<=10; ++t){
    for(n=0; n<10; ++n){
      if((p=n+x)>=10) p-=10;
      DNA[t][n]=DNA[1][p];
    }
    ++x;
  }
  random(t,DNA);
}
function random(t,tgt,  n){
  for(; t<=P; ++t){
    for(n=0; n<10; ++n) tgt[t][n]=tgt[1][n];
    for(n=0; n<10; ++n) swap(tgt[t],n,rnd(10));
  }
}
function swap(a,p1,p2,  tmp){ tmp=a[p1]; a[p1]=a[p2]; a[p2]=tmp; }

# 結果の表示
function Result(  t,n,x){
  if(GeneSize>10) exit;

  printf("¥n[%d]¥n", G);
  for(t=1; t<=P; ++t){
    printf("%2d: ", t);
    for(n=0; n<GeneSize; ++n)
      printf("%s:%d ", Use[n], DNA[Idx[t]][n]);
    printf("(%d)", Eval[Idx[t]]);
    display(DNA[Idx[t]]);
  }
}
function display(tgt,  n){
  printf("  %d", calc(tgt, WORD[1]));
  for(n=2; n<WordCnt; ++n) printf("+%d", calc(tgt, WORD[n]));
  printf("[=%d]", calcTerm(tgt));
  printf("=%d¥n", calcAns(tgt));
}
```

```
#
# 適応度の算出
#
function Fitness(tgt,  ev,term,ans,t,n,st,sa){
  term=calcTerm(tgt);
  ans=calcAns(tgt);
  t=wrongZero(tgt);
  ev=(TopCnt-t)*100;
  if(t==0 && term==ans) ev+=1000;
  st=term "";
  sa=ans "";
  if(length(st)==length(sa)){
    dif=0;
    for(n=1; n<=length(st); ++n){
      if(substr(st,n,1)!=substr(sa,n,1)) ++dif;
    }
    ev += int((length(st)-dif)/length(st)*100);
  }else{
    dif=(term>ans)?(ans/term):(term/ans);
    ev += int(dif*10);
  }
  return ev;
}
function calcTerm(tgt,  n,term){
  for(n=1; n<WordCnt; ++n) term+=calc(tgt,WORD[n]);
  return term;
}
function calcAns(tgt){
  return calc(tgt,WORD[WordCnt]);
}
function calc(tgt,src,  sz){
  if((sz=length(src))==0) return 0;
  return (calc(tgt,substr(src,1,sz-1))*10 + tgt[Cnv[substr(src,sz)]]);
}
function wrongZero(tgt,  n,t){
  for(n=0; n<TopCnt; ++n) if(tgt[Cnv[Top[n]]]==0) ++t;
  return t;
}
function abs(n){ return ((n>=0)?(n):(-n)); }

#
# 終了判定
```

```
#
function Termination(){
  if(MAX<Eval[Idx[1]]){
    if(Eval[Idx[1]]>1000) printf("*"); else printf(" ");
    if(prevMAX<MAX) prevMAX=MAX;
    MAX=Eval[Idx[1]];
    printf("%-11s", sprintf("[%d](%d)", G, MAX));
    display(DNA[Idx[1]]);
  }
  return (G>=300);
}


# 再生 (※次世代を現世代の配列にコピー)
function Regeneration(dst,src,t,  n){
  for(n=0; n<10; ++n) dst[t][n]=src[t][n];
}


#
# 選択
#
function Selection(nxDNA,  t,x,n,n1,n2){
  for(t=1; t<=int(P*0.2); ++t){
    ++x;
    for(n=0; n<10; ++n) nxDNA[x][n]=DNA[Idx[t]][n];
  }
  if(x>0 && ((P-x)%2)){
    ++x;
    for(n=0; n<10; ++n) nxDNA[x][n]=nxDNA[1][n];
  }
  for(t=x+1; t<int(P*0.8); t+=2){
    n1=Idx[rnd(int(P*0.3))+1];
    do{ n2=Idx[rnd(P)+1]; }while(n1==n2);
    @rz_Crossover(n1, n2, DNA, t, t+1, nxDNA);
  }
  random(t,nxDNA);
}

#
# 交叉
#
```

```
function Crossover(p1,p2,src,c1,c2,dst,  cut){
  if(rnd(100)>=Cr){
    for(n=0; n<10; ++n){
      dst[c1][n]=src[p1][n];
      dst[c2][n]=src[p2][n];
    }
    return;
  }

  cut=rnd(GeneSize-1)+1;
  if(rnd(100)>=50) cross(p1,p2,src,c1,c2,dst,   0,cut, cut);
  else             cross(p1,p2,src,c1,c2,dst, cut, 10,   0);
}
function cross(p1,p2,src,c1,c2,dst, st,ed, st2,  n,list1,list2,p,x){
  delete list1;
  delete list2;
  for(n=st; n<ed; ++n){
    list1[(dst[c1][n]=src[p1][n])]=1;
    list2[(dst[c2][n]=src[p2][n])]=1;
  }
  n=st2;
  for(p=0; p<10; ++p){ x=src[p2][p];  if(!(x in list1)) dst[c1][n++]=x; }
  n=st2;
  for(p=0; p<10; ++p){ x=src[p1][p];  if(!(x in list2)) dst[c2][n++]=x; }
}

#
# 突然変異
#
function Mutation(t,src,  n,p,x){
  if(t<3) return;

  for(n=0; n<10; ++n)
    if(rnd(100)<Mr) swap(src[t],n,rnd(10));
  return;
}
```

---

7. 動作環境について

執筆時の動作確認は、
GNU Awk 4.0.0

で、WinXP の DOS 窓で行なってます。(ちなみに Cygwin 版バイナリです)

先にも書きましたが今回のスクリプトは gawk4.0.0 独自の機能を意識して使っていますので他の awk で動くようにするためには、いくつか手直しが必要です。

・"@include"を使って、別の awk スクリプトファイルを取り込んでます。
・配列の配列を使ってます。(ex. a[1][2])
・Indirect ファンクション呼び出しを使ってます。(ex. @func())
・独自に定義した比較関数が 3 番目の引数に指定できる asort()を使ってます。

8. その他

本記事のサンプルスクリプトはフリーソフトです。
著作権は作者であるＹさにあります。
ただし転載、再配布、改造、消去は自由です。
(できましたら素晴らしい改造を加えた後に TSC 編集委員会宛に投稿してくださいませ)
また、このソフトを使用した事による損害が発生したとしても、損害に対しては一切の責任を負いかねます。

(投稿: 2011 年 10 月 22 日)

追録の加除

ムムリク

加除式本

加除式本（加除本）という本があります。多くの人にとってはあまりなじみのない本ですが、公務員であったり法曹関係者にはなじみのある本です。企業でも総務関係の部署では扱っているところがあるかもしれません。法令や例規といったものがまとめられた本で、多くはその関係するところ専門の内容だけをまとめたものになっています。

一般によく知られる六法全書といったものとはやや異なり、たとえば警察であったり金融機関であったりといった業務に特化した内容だけをまとめています。このためその業務に従事しない多くの人々には、まず目にすることのない本です。

加除式という意味はページの入れ替えができるということです。法律は常に修正が行われたり新しく作られています。このため通常の六法全書などでは年度版などでそれらに対応します。しかし本をまるまる作り直すということになるために無駄も生じますし、変更にたいして即応することが難しいのが欠点です。

そこで変更のあった部分のページだけを印刷し、それを入れ替えのための指示書（加除表・加除参考表などという）にそって入れ替えることで、常に最新の状態を維持することを可能にしています。本の形態としてはルーズリーフ式のノートを連想すれば概ねあっています。

追録

この入れ替えのためのページをまとめたものを追録（ついろく）といいます。年間の発行回数などはそれぞれの本によって異なります。一回のページ数もその時々の修正量によって変わってくるので一様ではありません。変更が多ければページ数の多い追録が発行されることになります。

加除作業

作業になれてしまえば通常一回の加除作業は 5 分から 10 分程度のものが多いですが、量によっては数十分から一時間あまりかかることもあります。さらに、作業を怠って追録がたまっていたりして大量に作業をしなくてはならないようになると数時間に及ぶこともあります。極端な場合数日をかけて行うようなことすらありえます。

多くの場合その都度作業しておけばわずかな時間ですが、業務の合間に自分で行うということになるため後回しにしてしまうということが往々にしておきます。それが度重なる

と追録の数がどんどんと増えて、いざ作業しようと思ってもその分量の前にますます意欲をなくしてしまうことにもなります。

加除作業の支援

ところで、こうして入れ替えが必要になる部分というのは割りと同じ部分であったりすることがままあります。このため順序よく作業をしていると同じあたりのページを入れ替えてはまた除く、という作業を繰り返すようになります。つまり、一番はじめの追録ページは最終的には不要であるかもしれないということです。

もちろんこれは追録をたくさん溜め込んでしまったような場合だからこそいえることではあるのですが、そのような場合に最終的にどのような形になればいいのかがわかれば、あるいはその手間は軽減されるかもしれません。

そこで、追録を加除するという発想が生まれます。最終的に多数の追録のどの部分が残るのかがわかることで、まったく不要になる部分はそもそも除外することもできます。ただ、追録が少ない状況ではあまり効果はなさそうです。むしろ素直に順番通り作業したほうが早く簡単かもしれません。

また、仮に大量の追録がある場合でも、それぞれの追録から必要な部分を抜き取ったりという準備は却って煩雑になるかもしれません。このあたりが現実に導入に踏み切りにくい問題のひとつともいえそうです。

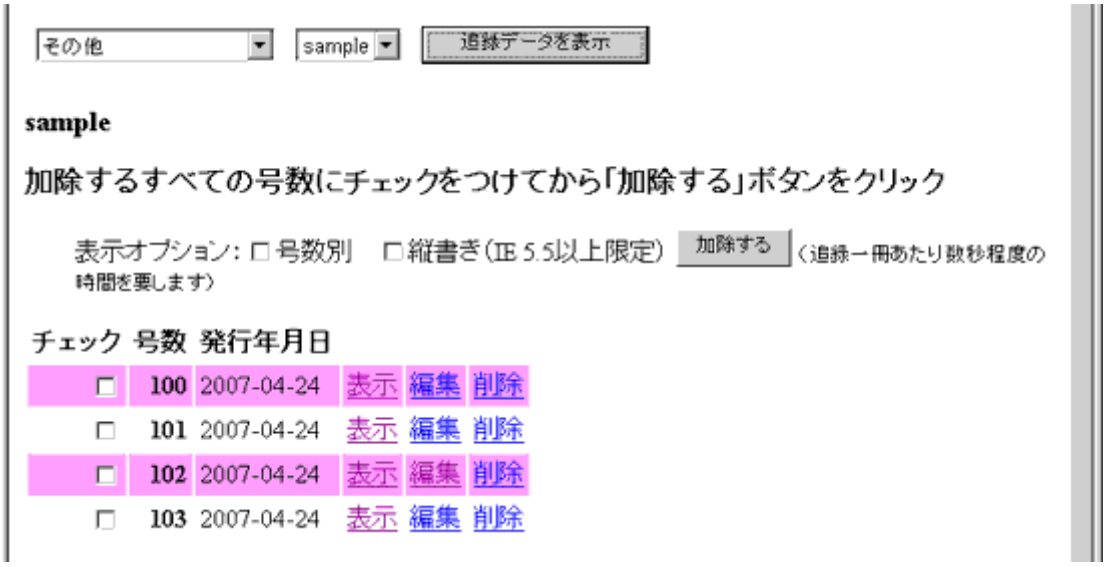
そのような意味からは電子書籍化されたほうが、加除という仕組みそのものはもっと活かしやすくなる可能性は高いのかもしれません。

とりあえずはできることを試してみようというのがこのプログラムです。（基本的には出版社サイドで利用価値があるかもしれないというプログラムです）

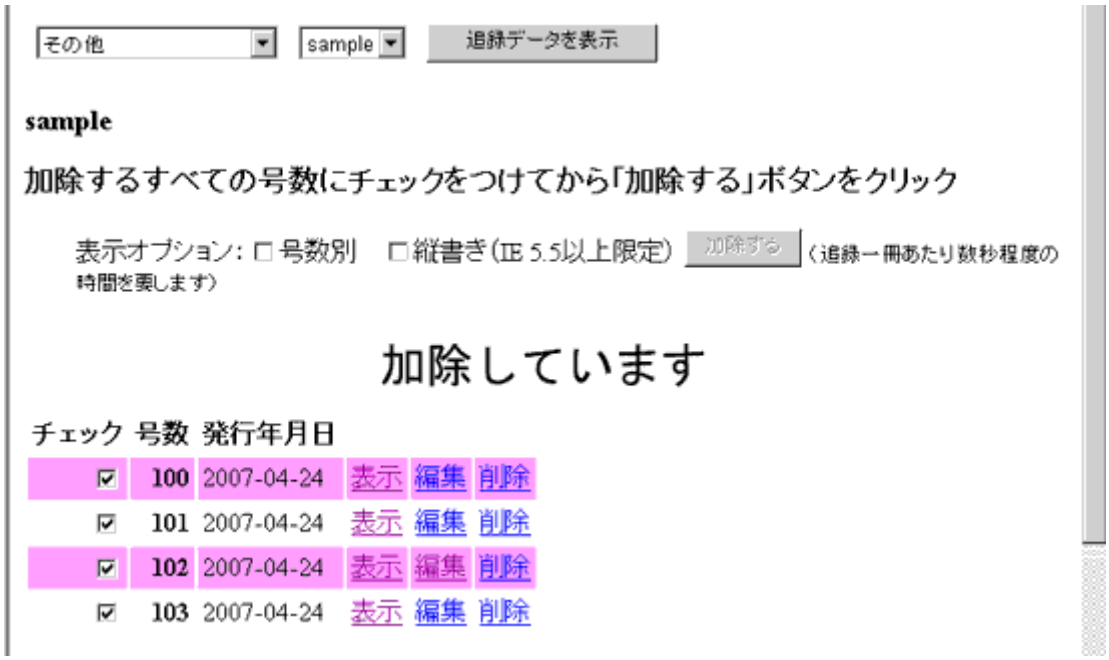
プログラムは Ruby on Rails で作られています。データベースに追録の情報を収めます。必要な追録の情報を選んで、加除ボタンを押すとしばらくして結果を表示します。あくまでもテスト的なプログラムですが、その流れを画像で紹介します。

加除の流れ(1)

加除の流れ(2)



加除の流れ(3)



加除の流れ(4)

sample追録 第 100 号 加除参考表 (この追録を削除)

巻数	編別	種別	除くページ	除く枚数	加える枚数	加えるページ		
1	①		55 72	9	9	同	編集	削除
1	①		75 76	1	1	同	編集	削除
1	①		91 96 (～100)	3	3	同	編集	削除
1	①		123 123 の12 (～200)	6	11	123 123 の22 (～200)	編集	削除
1	①		351 354	2	2	同	編集	削除
1	①		601 622	11	11	同	編集	削除
除く枚数: 32 枚								

結果サンプル(1)

sample 100号 ～ 103 号の合成加除表 (所用時間: 2.443 秒)

巻数	編別	種別	号数	除くページ	枚数	枚数	加えるページ	
1	①		103	55 72	9	9	同	内包を削除
1	①		100	75 76	1	1	同	
1	①		100	91 96 (～100)	3	3	同	
1	①		100	123	1	1	同	枚数は計算上
1	①		102	123 の1 123 の21	7	7	同	
1	①		100	123 の22		1	123 の22 (～200)	枚数は計算上 加えるページのみ
1	①		100	351 354	2	2	同	
1	①		100	601 622	11	11	同	
除く枚数: 34 枚 行数:8 行 (元の除く枚数合計:59 枚 元の行数合計:11 行)								

結果サンプル(2)

sample 100号 ～ 103 号の合成加除表（所用時間: 2.463 秒）

巻数	1	1	1	1	1	1	1	1
編別	①	①	①	①	①	①	①	①
種別								
号数	100	100	100	102	100	100	100	103
除くページ	六〇一 六二二	三五一 三五四	一二三ノ二三	一二三ノ一 一二三ノ二二	一二三	九一 九六（～一〇〇）	七五 七六	五五 七二
枚数	11	2		7	1	3	1	9
枚数	11	2	1	7	1	3	1	9
加えるページ	同	同	一二三ノ二三（～一〇〇）	同	同	同	同	同
			枚数は計算加えるべー		枚数は計算			内包を削除

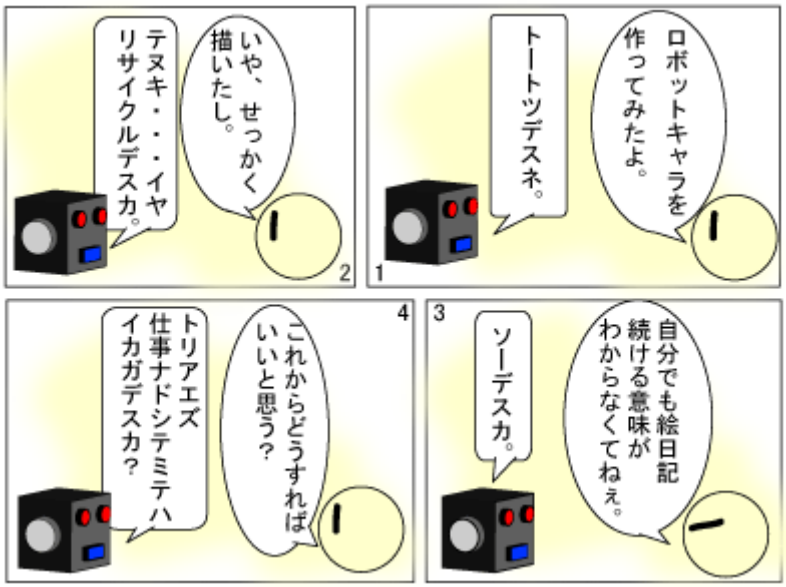
（投稿: 2011 年 11 月 10 日）

よしおさんとロボ太

海鳥

その１

「作ってみた」



手抜きキャラ出現。

jscripter より

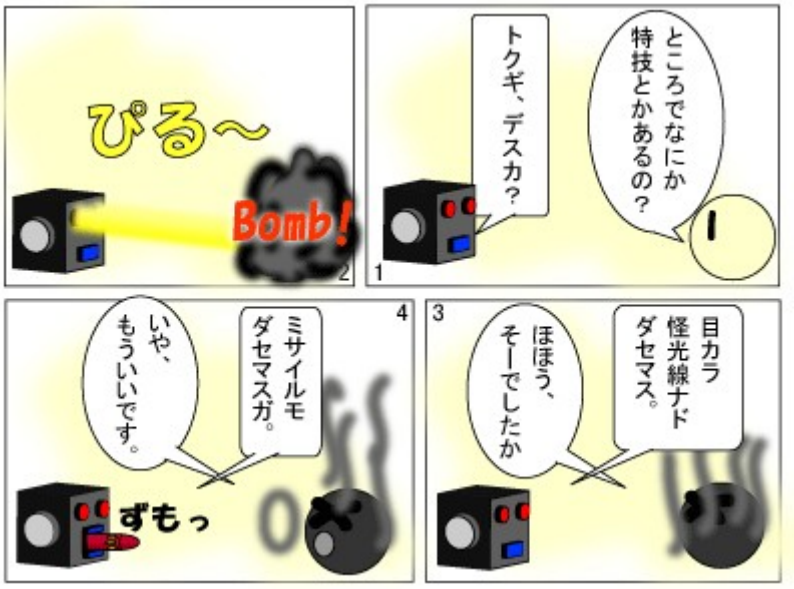
TSNET スクリプト通信第 13 号で「よしおさんとロボ太」掲載が無事終了した記念に、今号では全編一挙掲載して、楽しんでいただくことにしました。これまでの掲載は各作品が生まれた順序ではなかったので、本来の時系列に沿って鑑賞していただきたいからです。

よしおさんとロボ太

海鳥

その２

「特技」



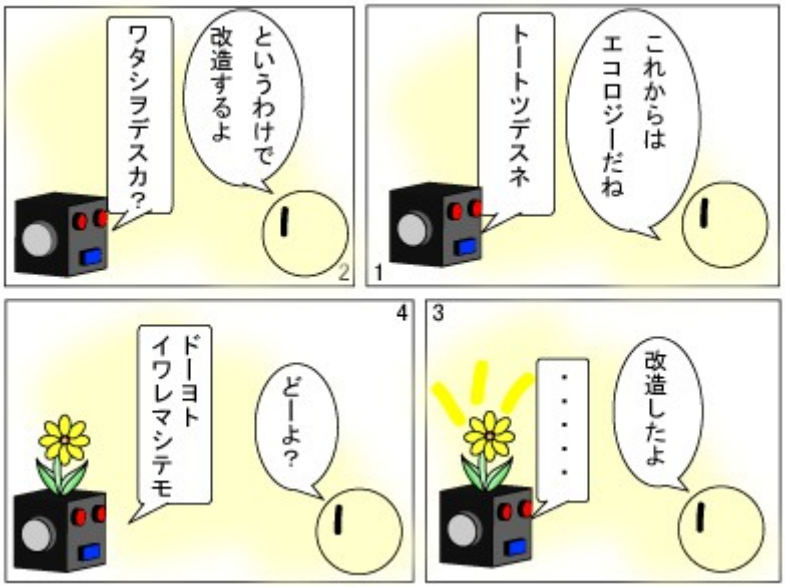
ミサイルの寸法がおかしい。

よしおさんとロボ太

海鳥

その３

「エコロとココロ(前編)」



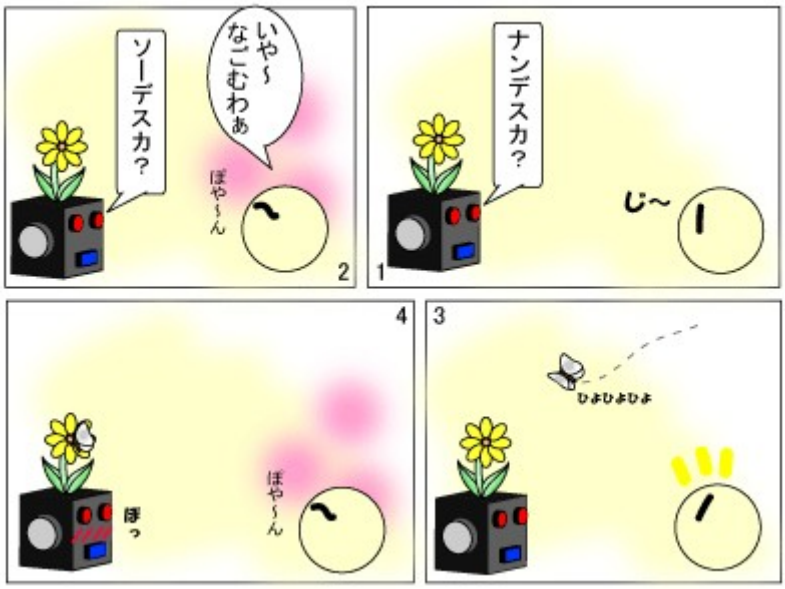
地球にやさしい(たぶん)

よしおさんとロボ太

海鳥

その４

「エコロとココロ(中編)」



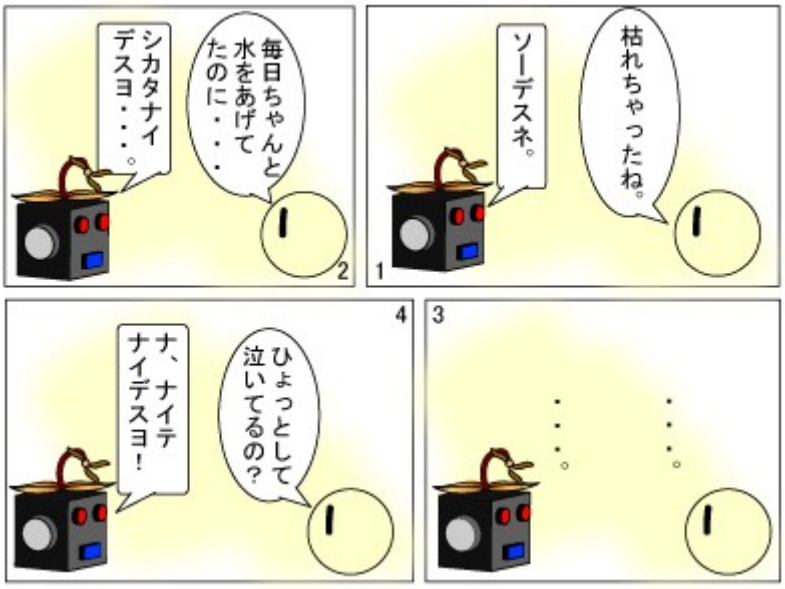
ツンデレロボ(違)

よしおさんとロボ太

海鳥

その５

「エコロとココロ(後編)」



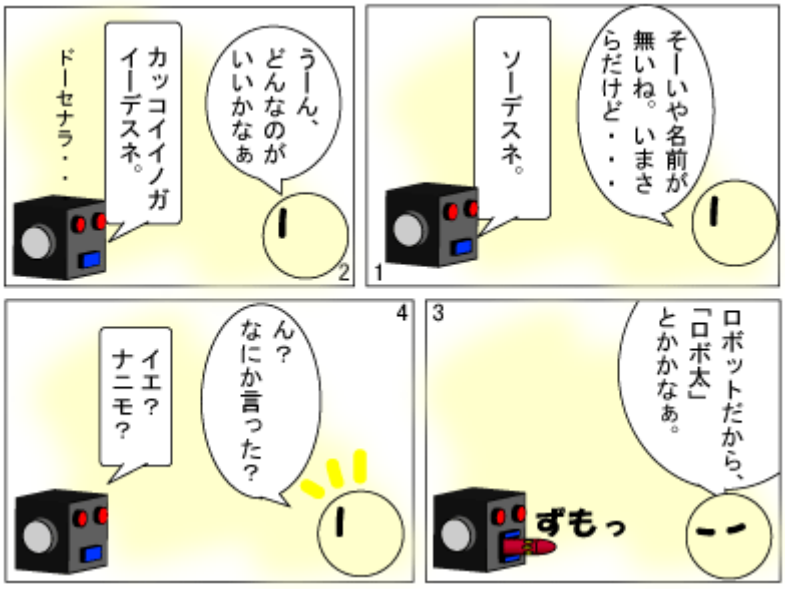
エコロとココロ。

よしおさんとロボ太

海鳥

その６

「命名」



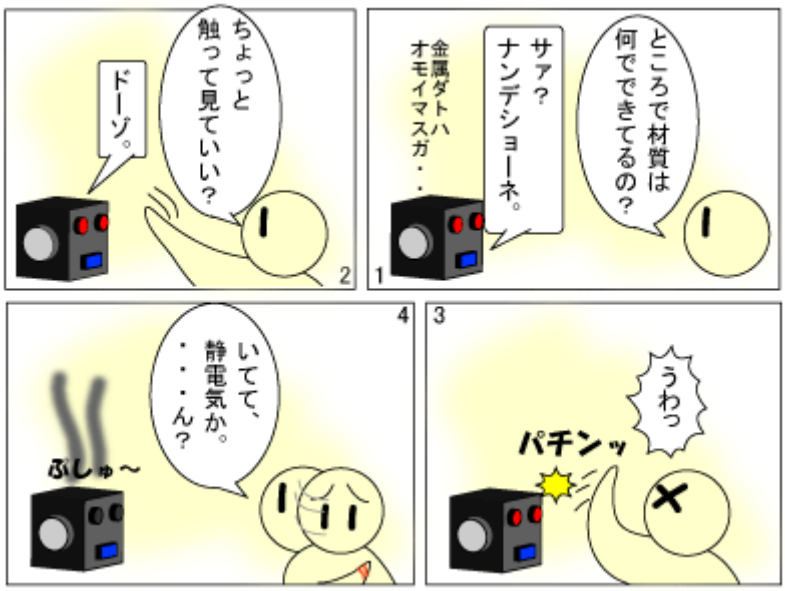
お気に召しませんでしたか？

よしおさんとロボ太

海鳥

その７

「乾燥注意報(前編)」



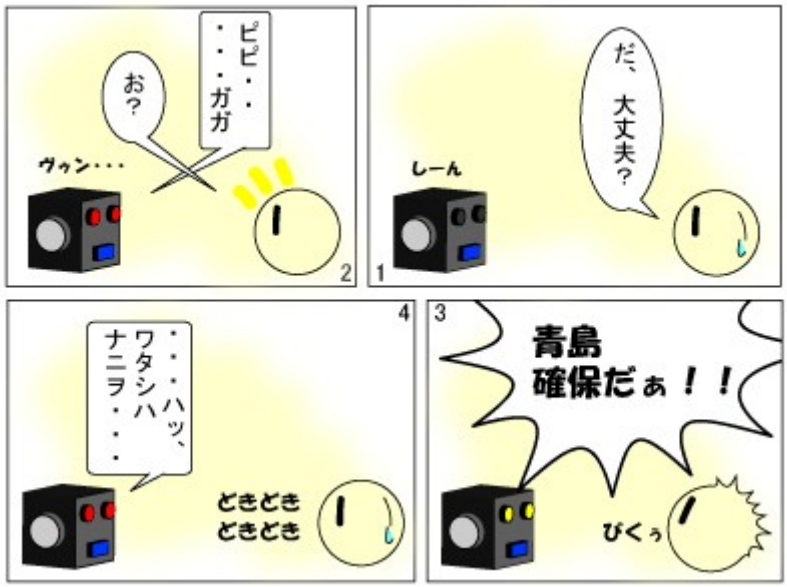
ロボ太？ロボ太ぁあああああああ！！

よしおさんとロボ太

海鳥

その８

「乾燥注意報(後編)」



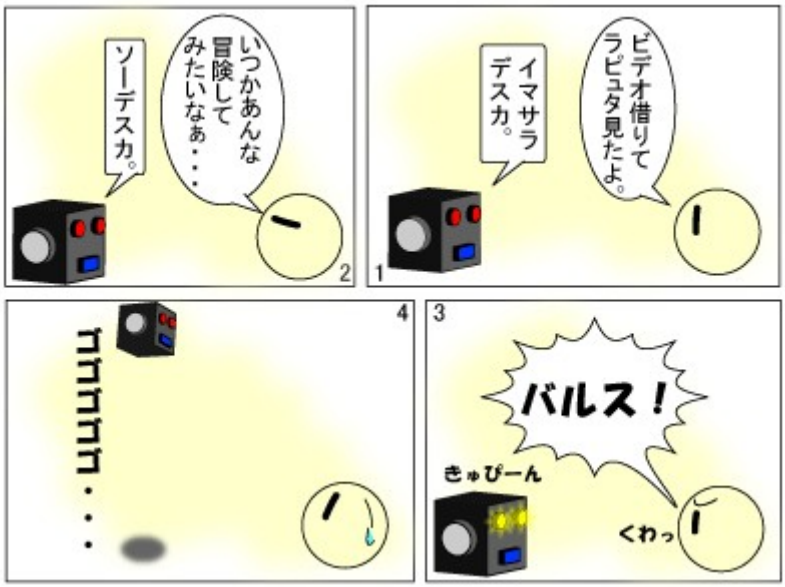
混信？

よしおさんとロボ太

海鳥

その９

「ラピ○タ」



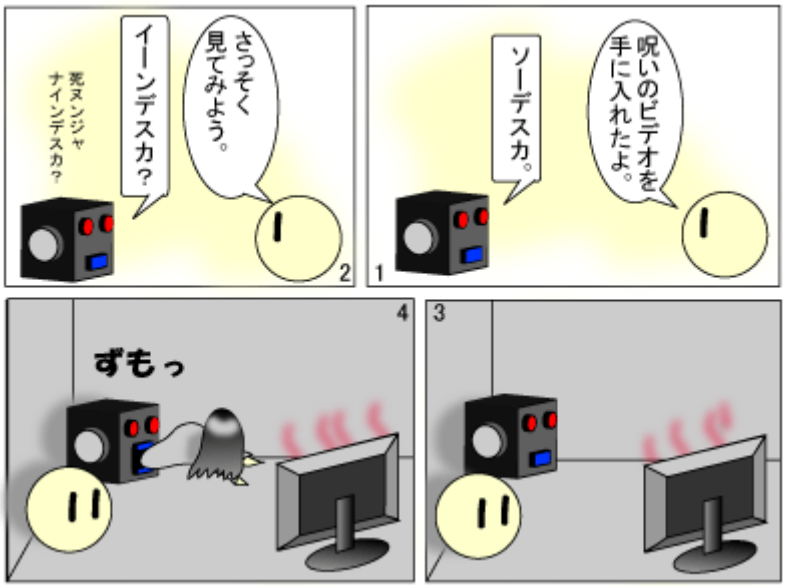
あのち～へい～せ～ん～♪ か～が～やく～の～は～♪

よしおさんとロボ太

海鳥

その 10

「ビデオ」



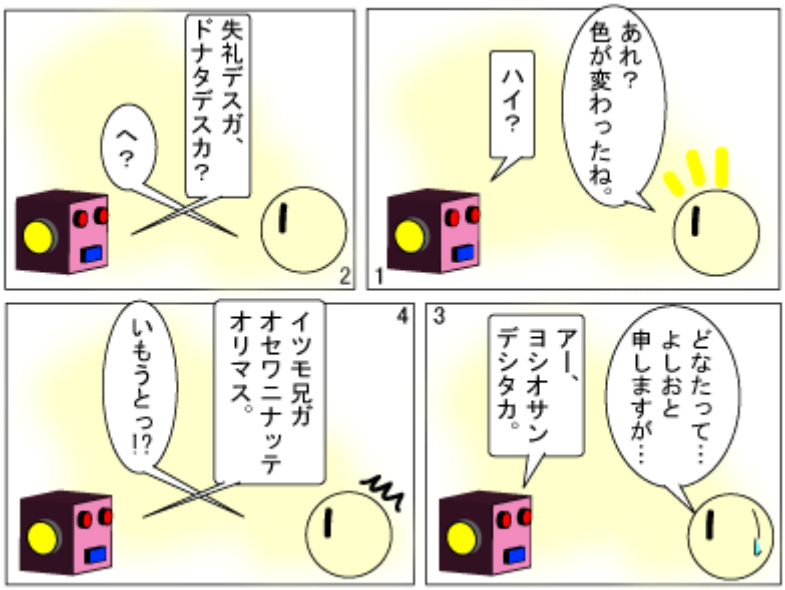
そこからっ！？

よしおさんとロボ太

海鳥

その 11

「色違い」



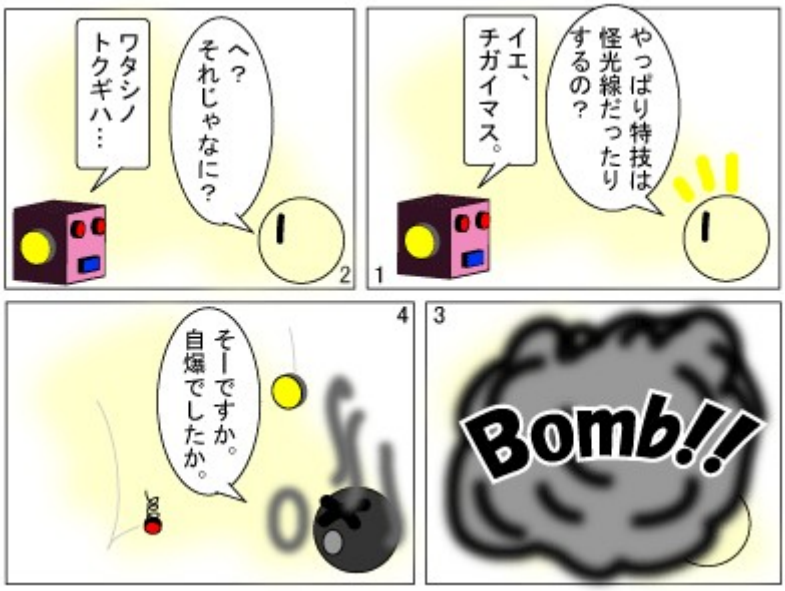
新キャラ出現(手抜きにも程がある)。

よしおさんとロボ太

海鳥

その 12

「続・特技」



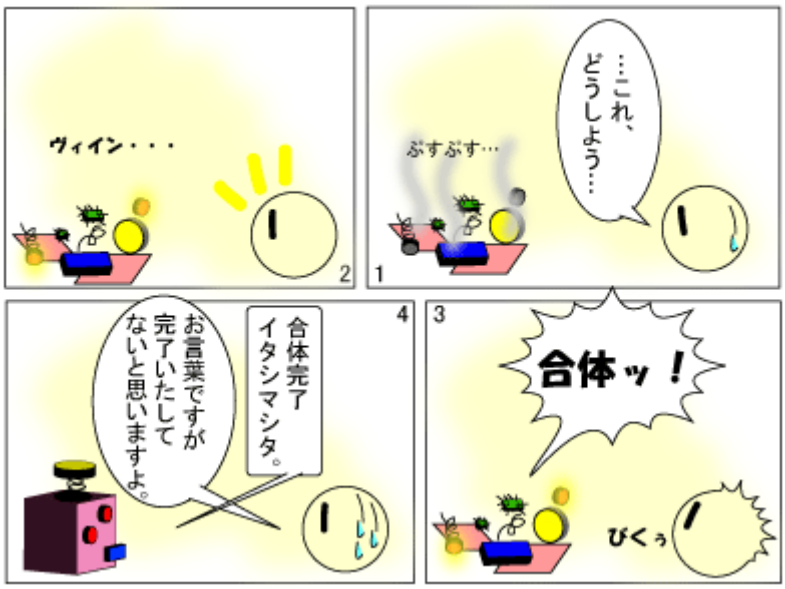
ロボ子(仮)？ロボ子ぉおおおおおおおおお！！！

よしおさんとロボ太

海鳥

その 13

「真・特技」



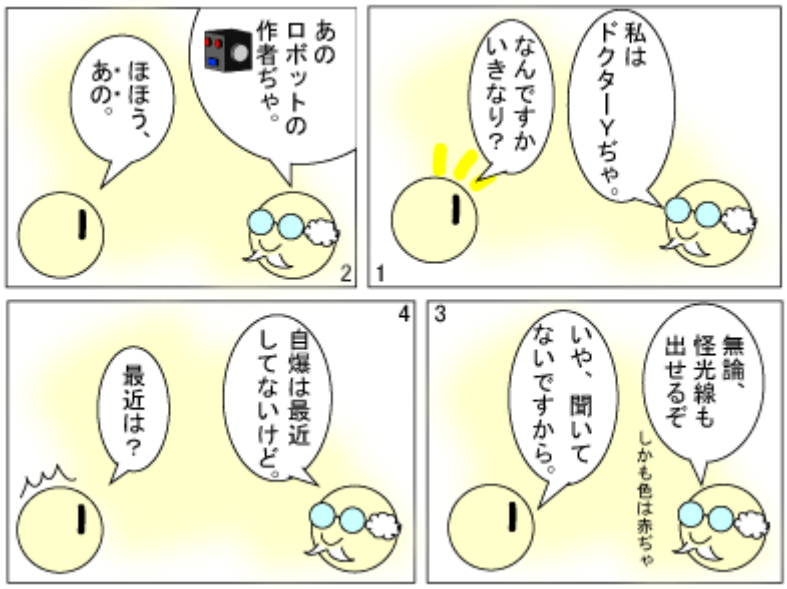
ピカソ？

よしおさんとロボ太

海鳥

その 14

「この親にして」



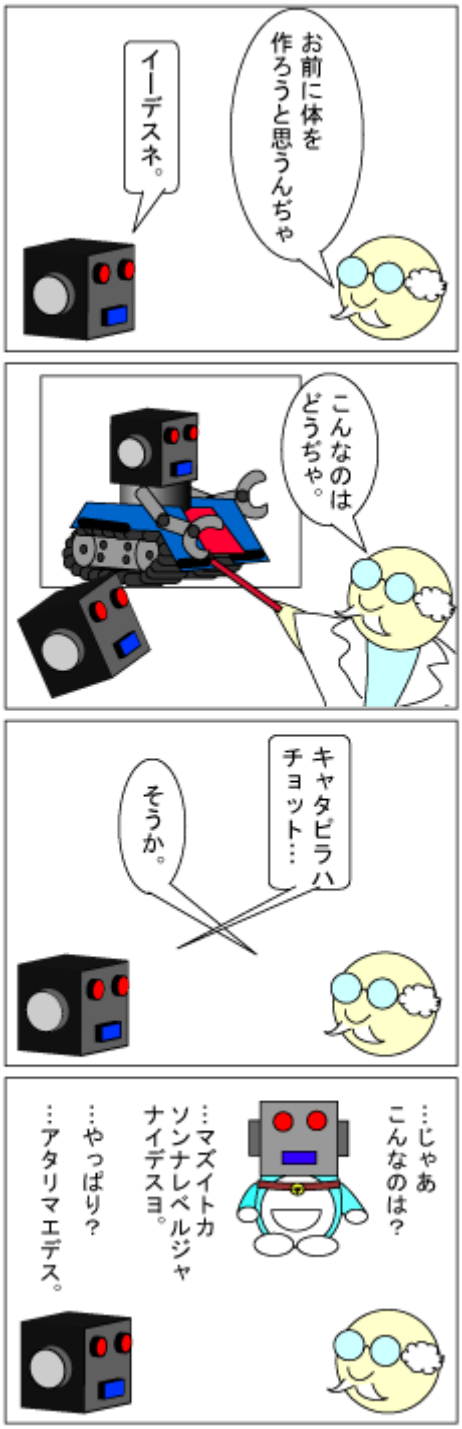
特技はあんたの趣味か。

よしおさんとロボ太

海鳥

その 15

「ロボに体を」



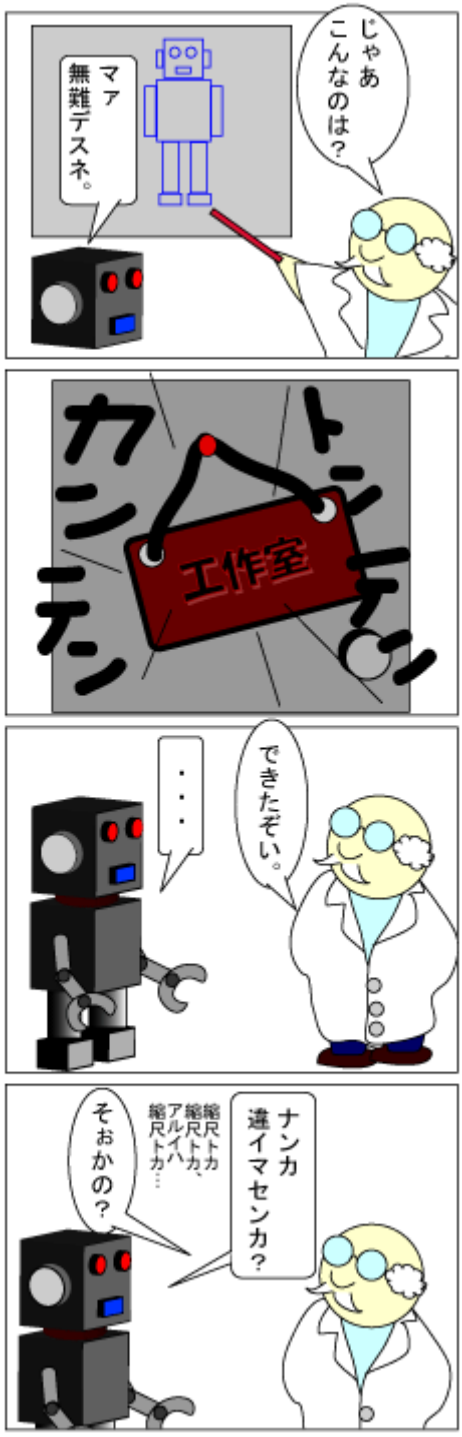
ついに手抜きをやめて体を描こうという作者の決意(挫折気味)。

よしおさんとロボ太

海鳥

その 16

「続・ロボに体を」



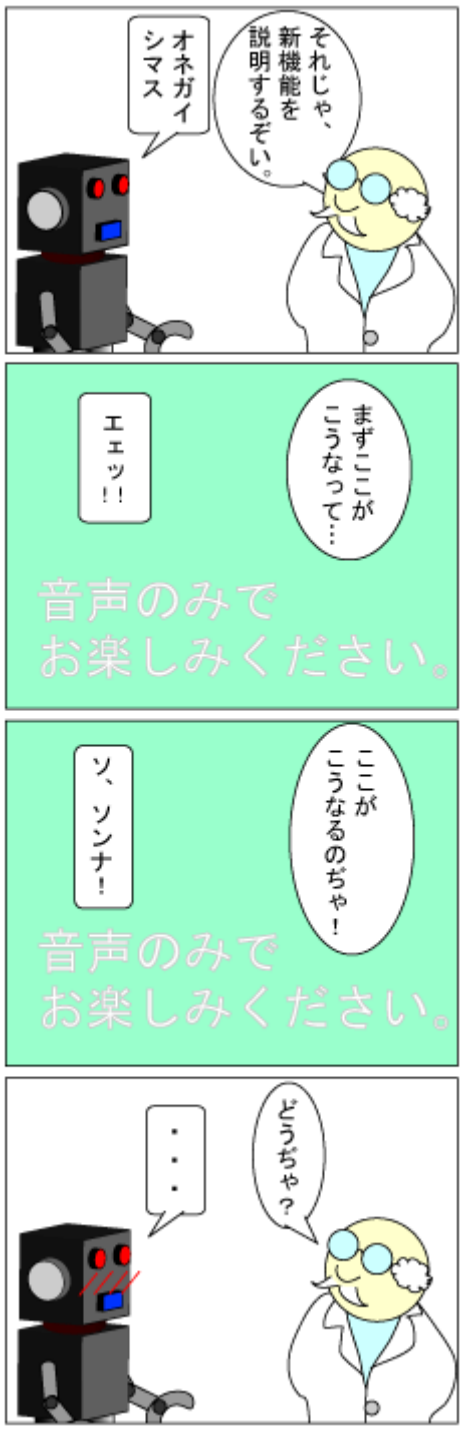
夏休みの工作レベル。

よしおさんとロボ太

海鳥

その 17

「秘・ロボに体を」



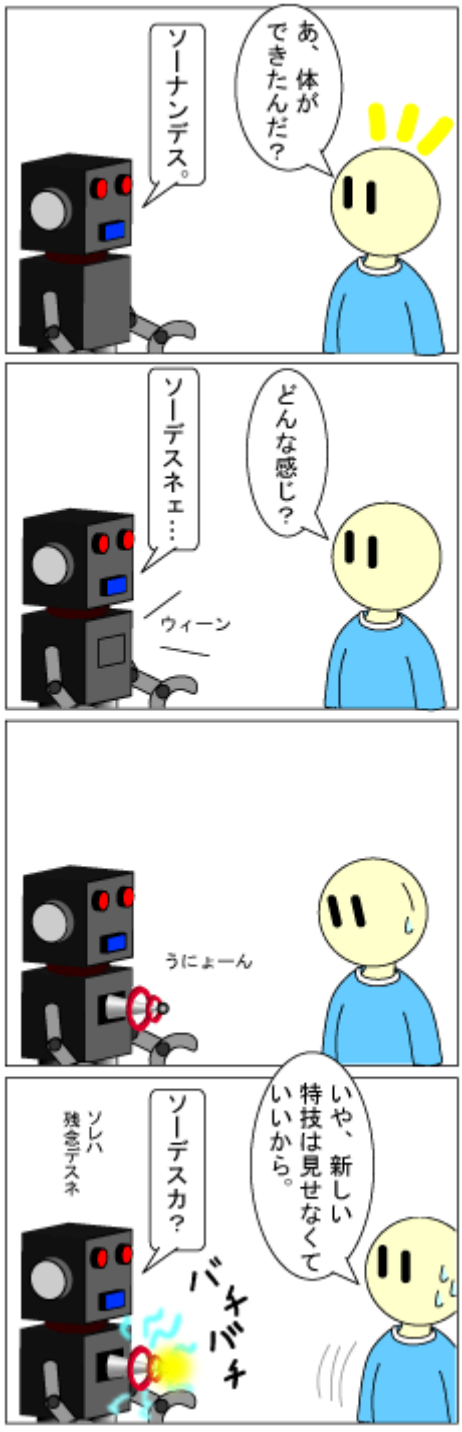
白衣と裸の二人があんなことやこんなことを(違)。

よしおさんとロボ太

海鳥

その 18

「激・ロボに体を」



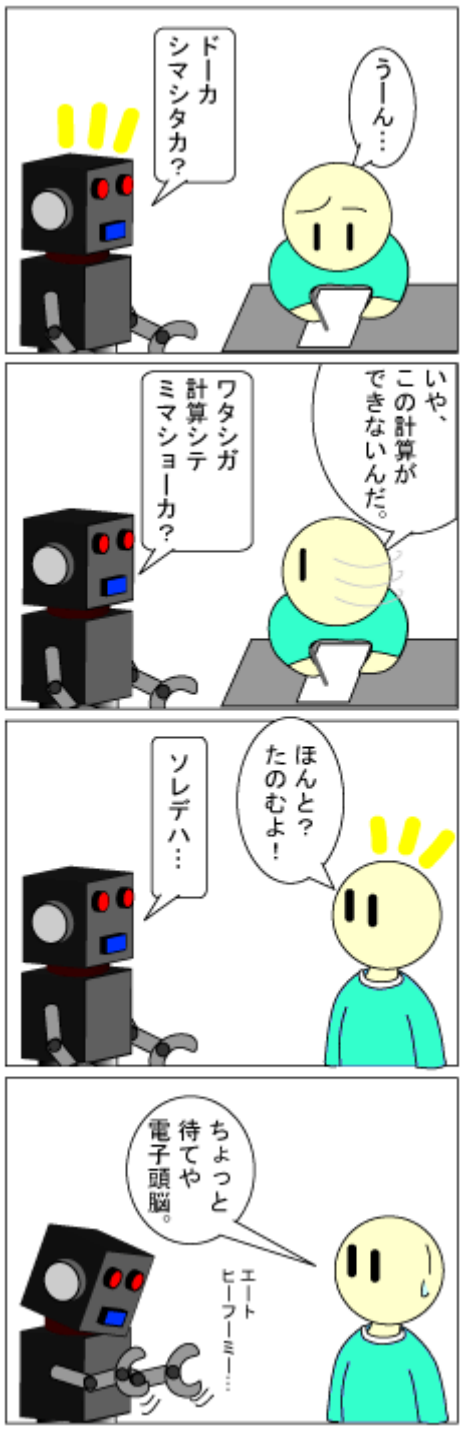
どんな特技だったんだろう・・・。

よしおさんとロボ太

海鳥

その 19

「高性能計算機」



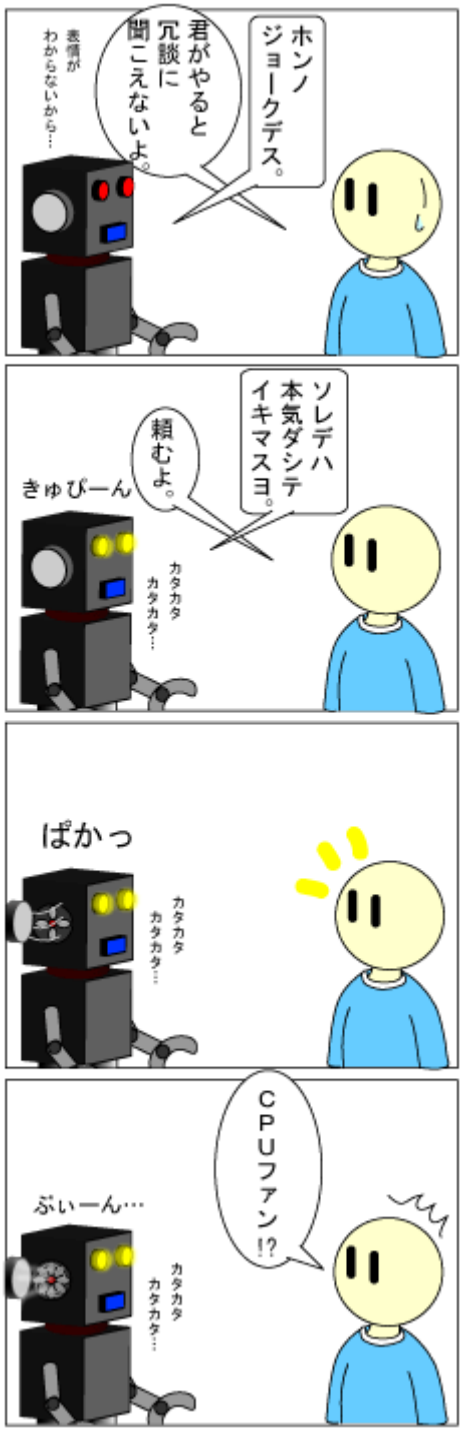
二進法？

よしおさんとロボ太

海鳥

その 20

「続・高性能計算機」



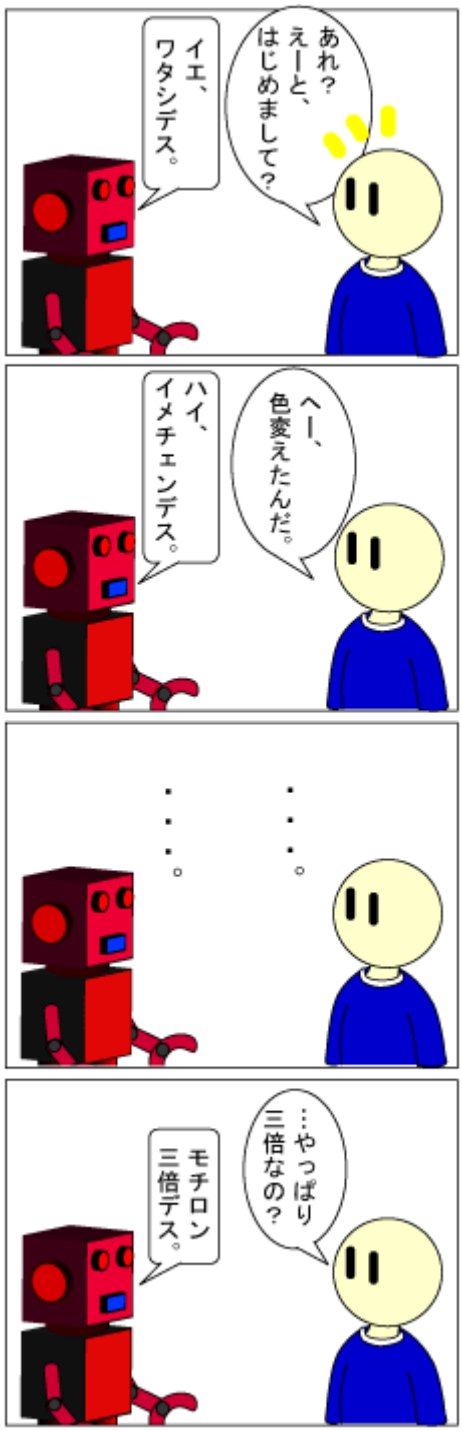
耳の構造がついに明らかに。

よしおさんとロボ太

海鳥

その 21

「専用」



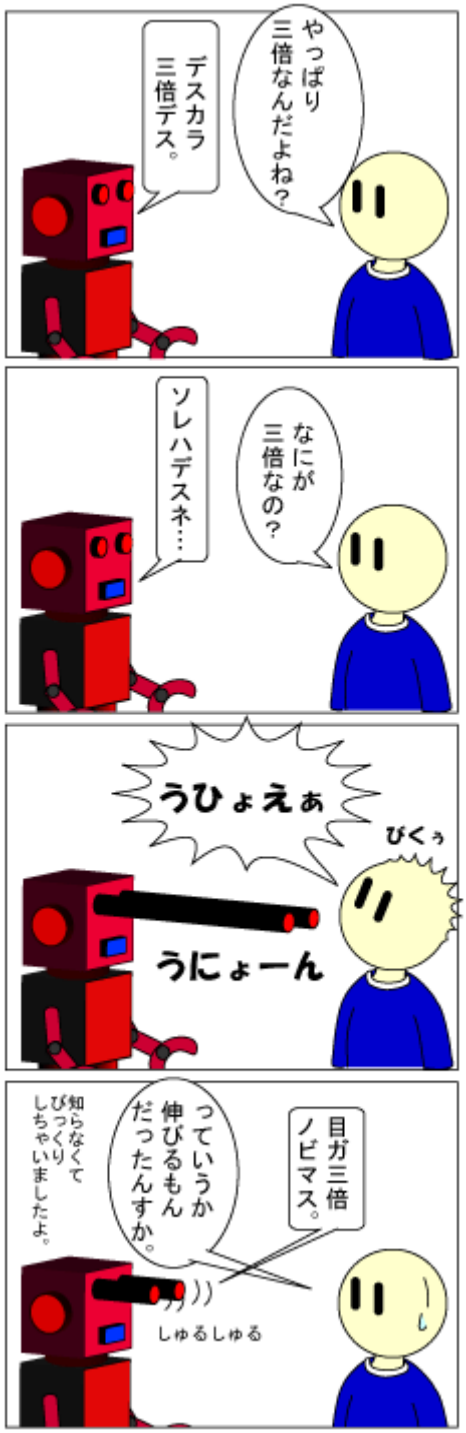
言い切っちゃったよ(さまあず三村風ツッコミ)。

よしおさんとロボ太

海鳥

その 22

「三倍」



何でもアリです(ロボットだから)。

よしおさんとロボ太

海鳥

その 23

「温・特技」

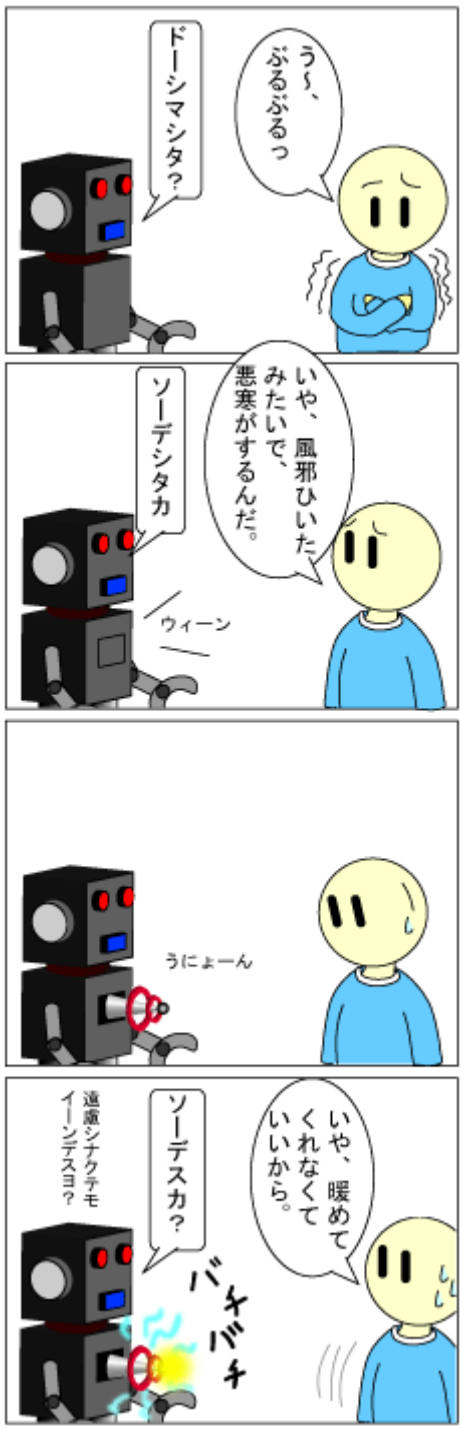
いや、便利だろうけれども。

よしおさんとロボ太

海鳥

その 24

「暖・特技」



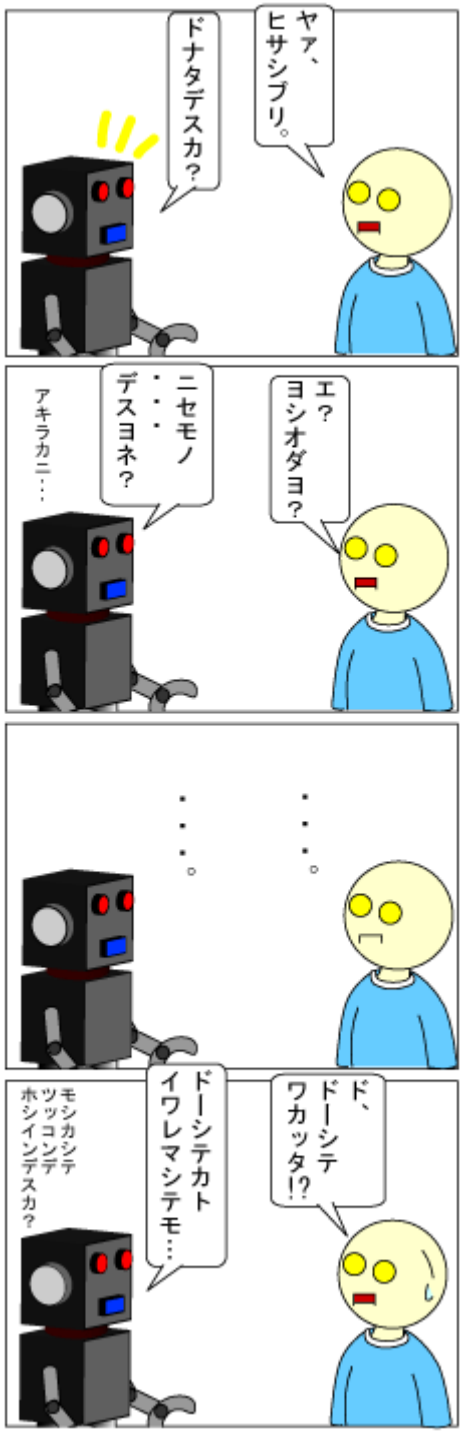
心も体もすぐポカポカ！(たぶん８０度くらいに)

よしおさんとロボ太

海鳥

その 25

「ヨシオサンー登場」



カタカナ会話。

よしおさんとロボ太

海鳥

その 26

「ヨシオサンー襲撃」

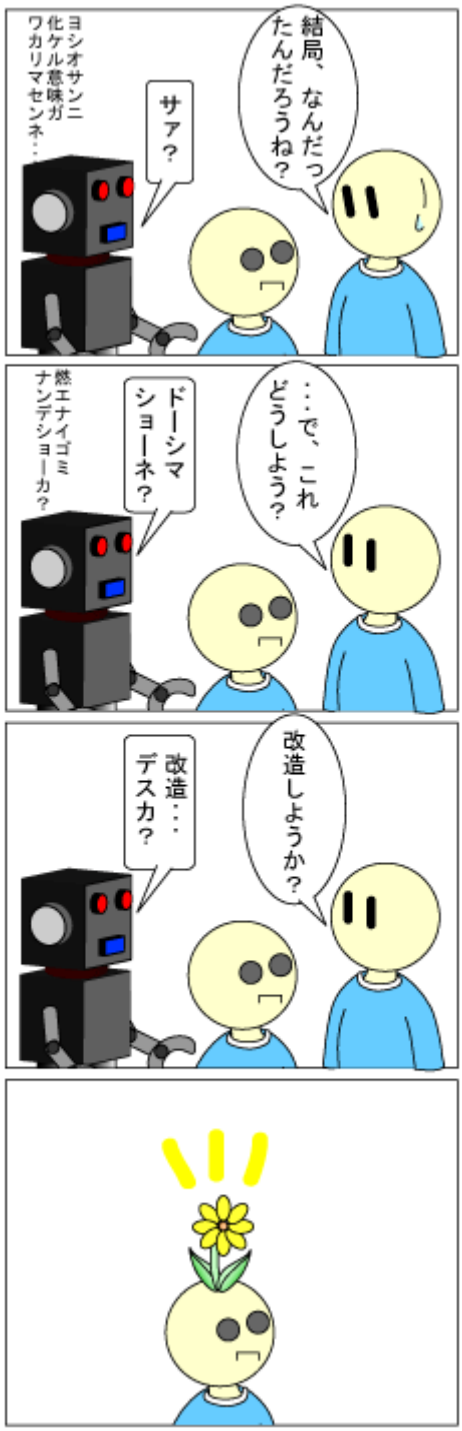
ドリル。

よしおさんとロボ太

海鳥

その 27

「ヨシオサン一転機」



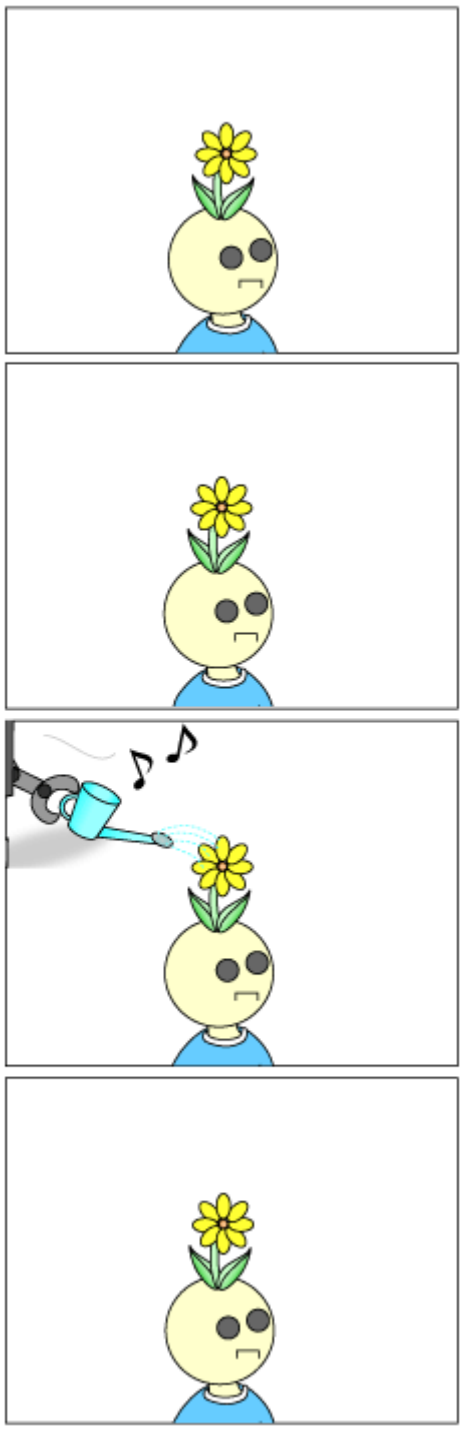
エコロボジー。

よしおさんとロボ太

海鳥

その 28

「ヨシオサンー余生」



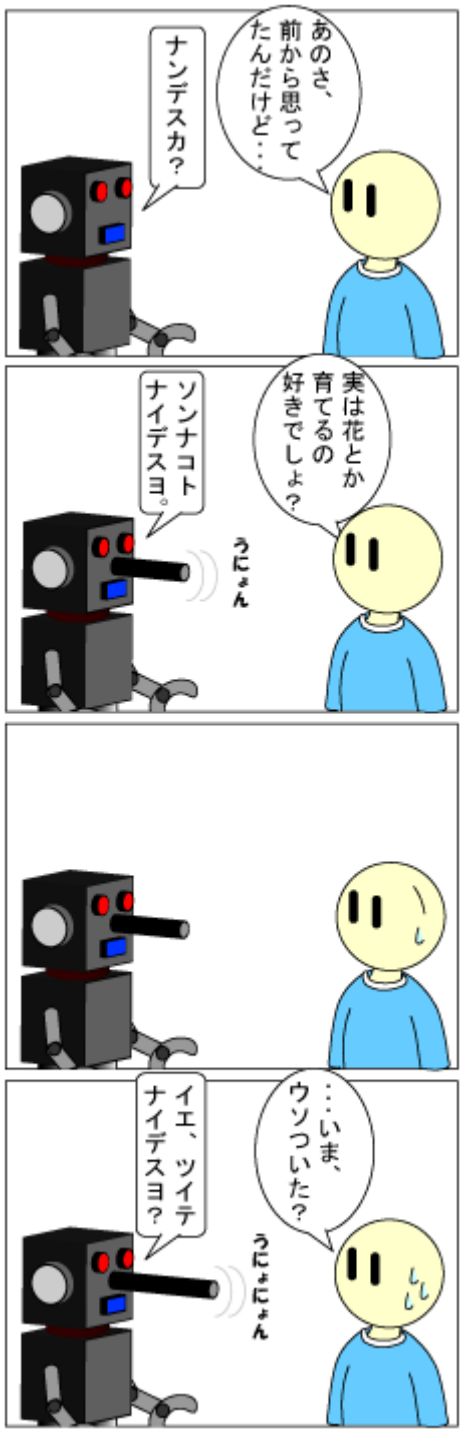
ガーデニングロボ。

よしおさんとロボ太

海鳥

その 29

「特殊機能」



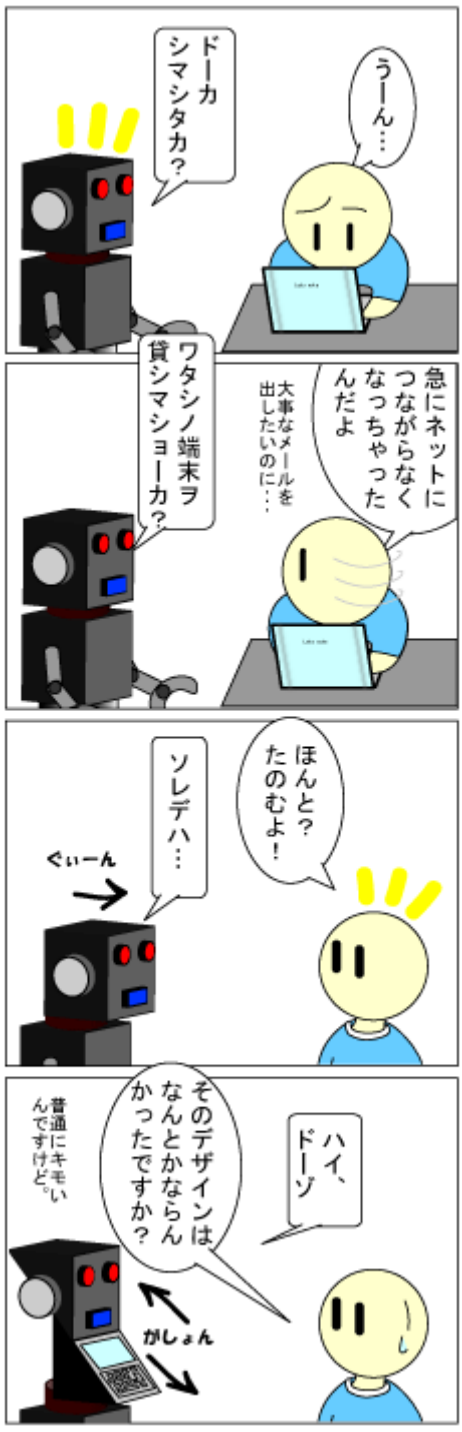
ピノ〇オ？

よしおさんとロボ太

海鳥

その 30

「変形ロボ」



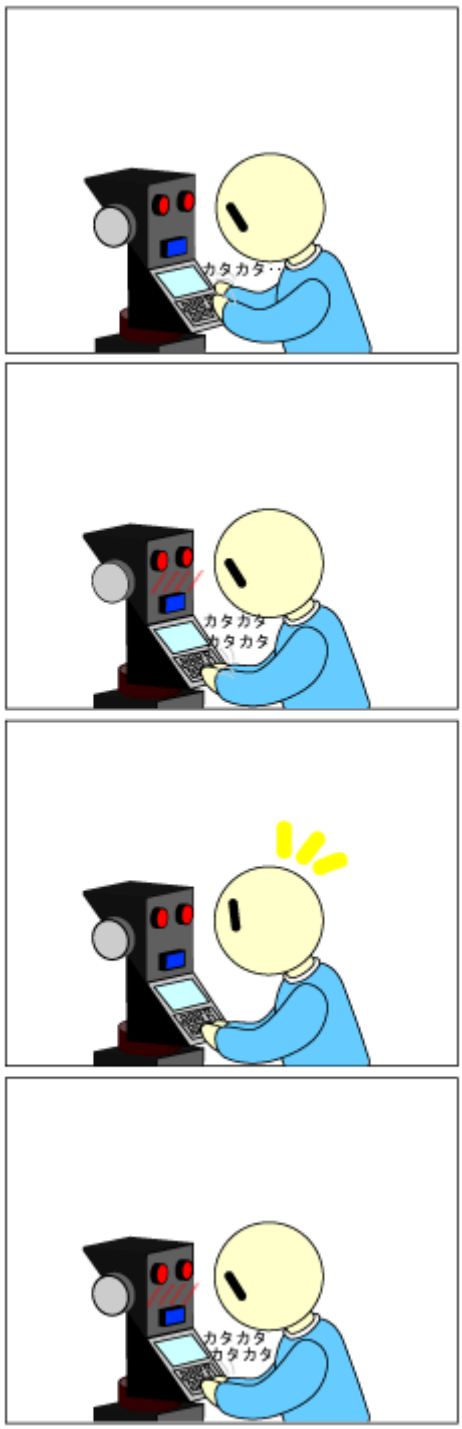
合体はしません(多分)。

よしおさんとロボ太

海鳥

その 31

「続・変形ロボ」



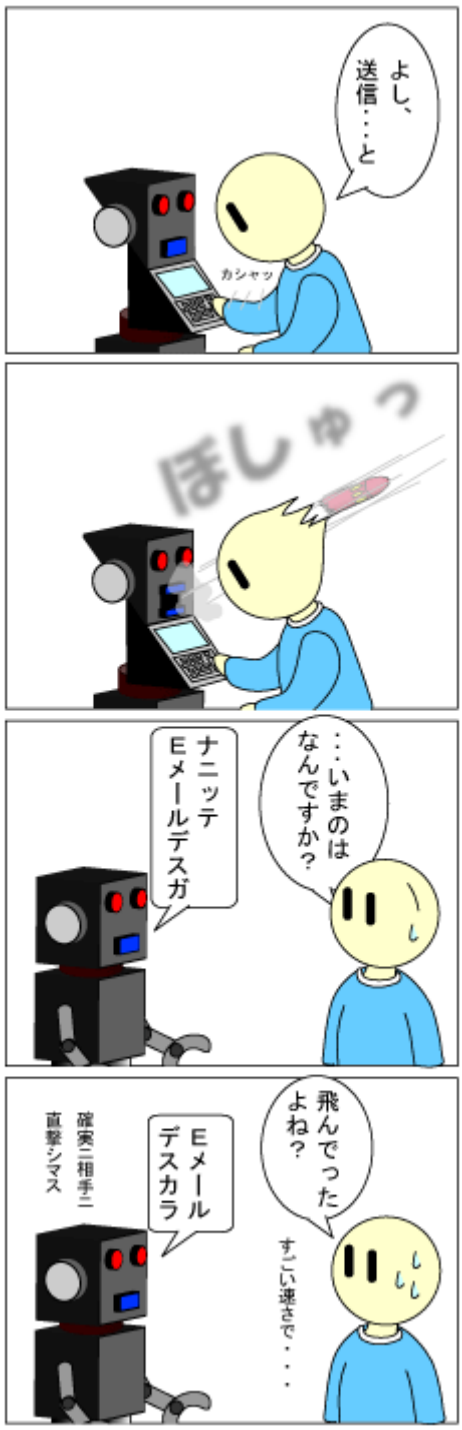
くすぐったいらしい。

よしおさんとロボ太

海鳥

その 32

「続々・変形ロボ」



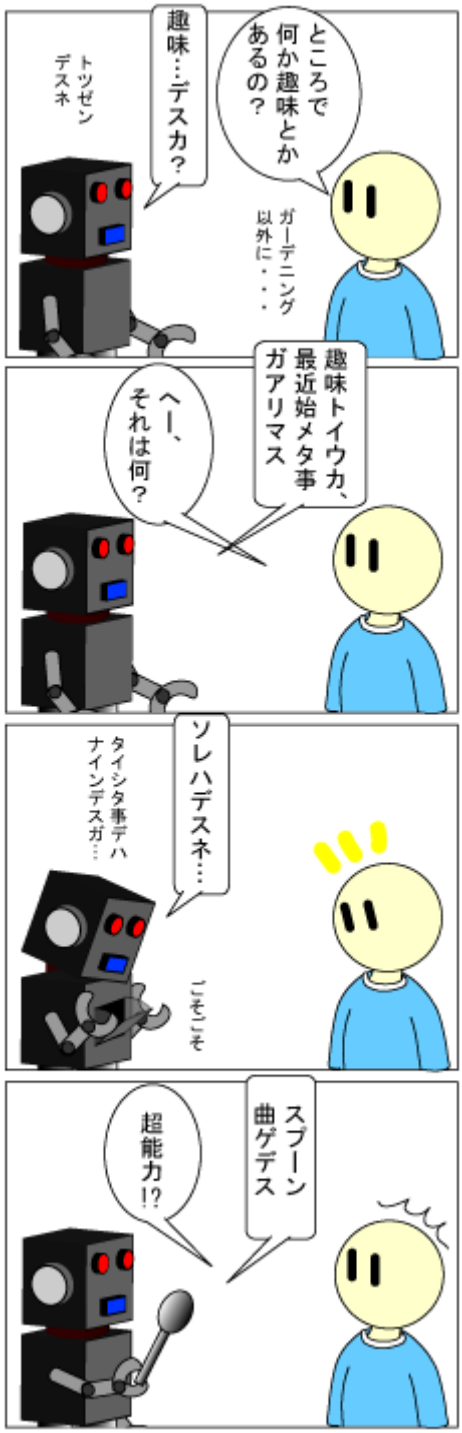
エマージェンシーメールの略？

よしおさんとロボ太

海鳥

その 33

「超科学」



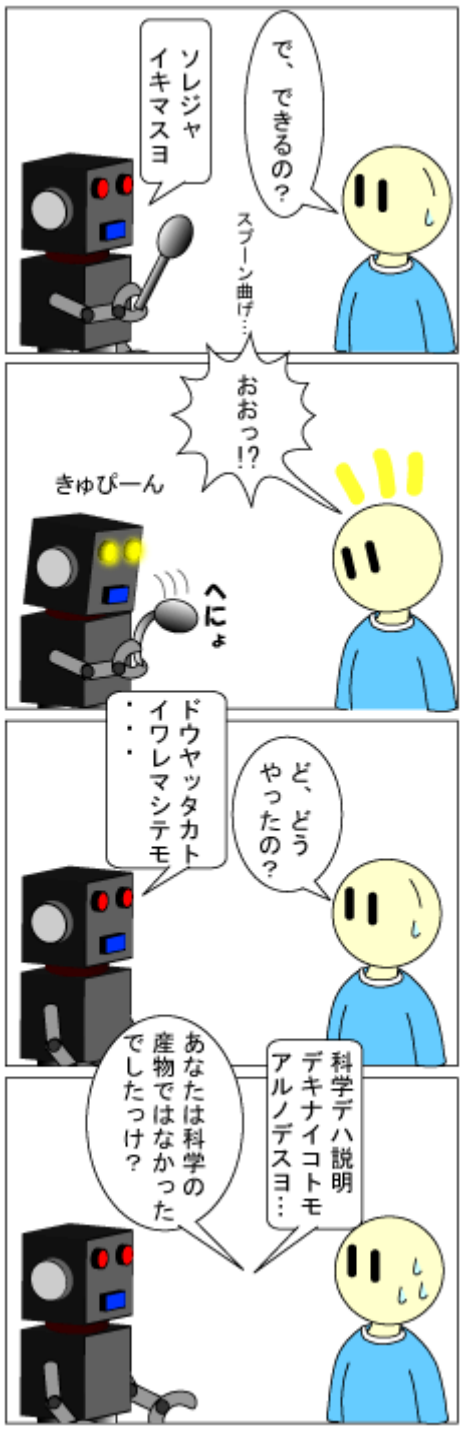
４〇元ポケット？

よしおさんとロボ太

海鳥

その 34

「続・超科学」



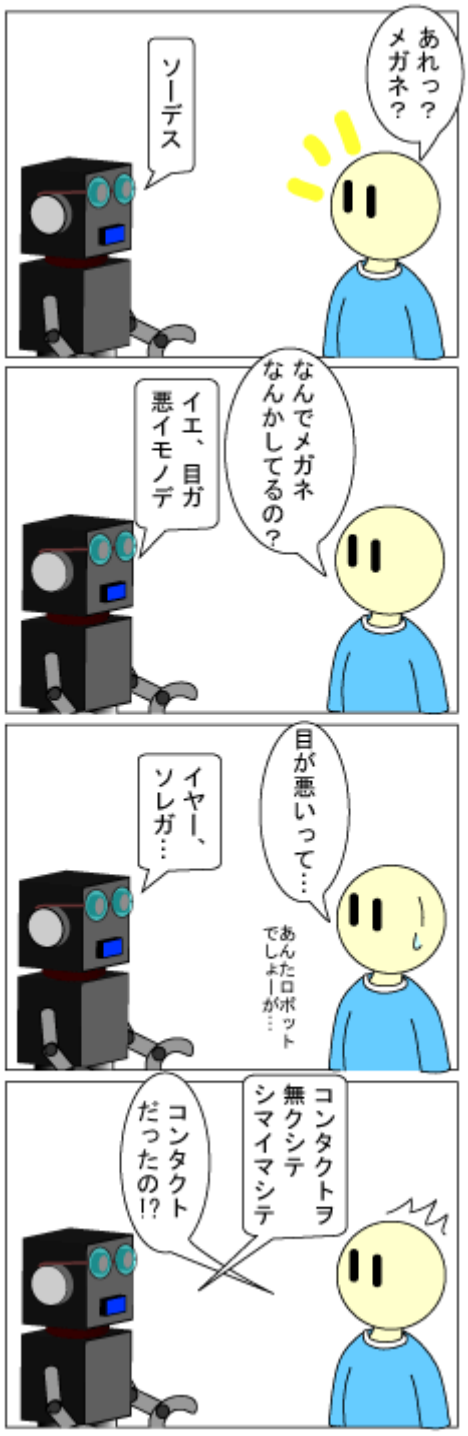
ロボ諭す

よしおさんとロボ太

海鳥

その 35

「メガネロボ」



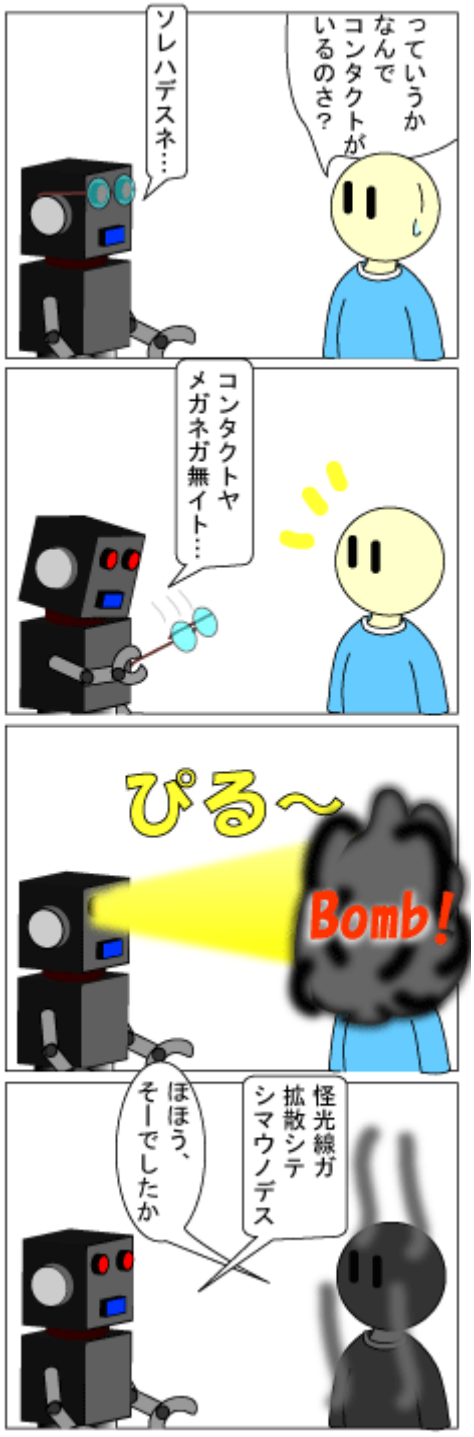
メガネっ子(言い切った)

よしおさんとロボ太

海鳥

その 36

「続・メガネロボ」



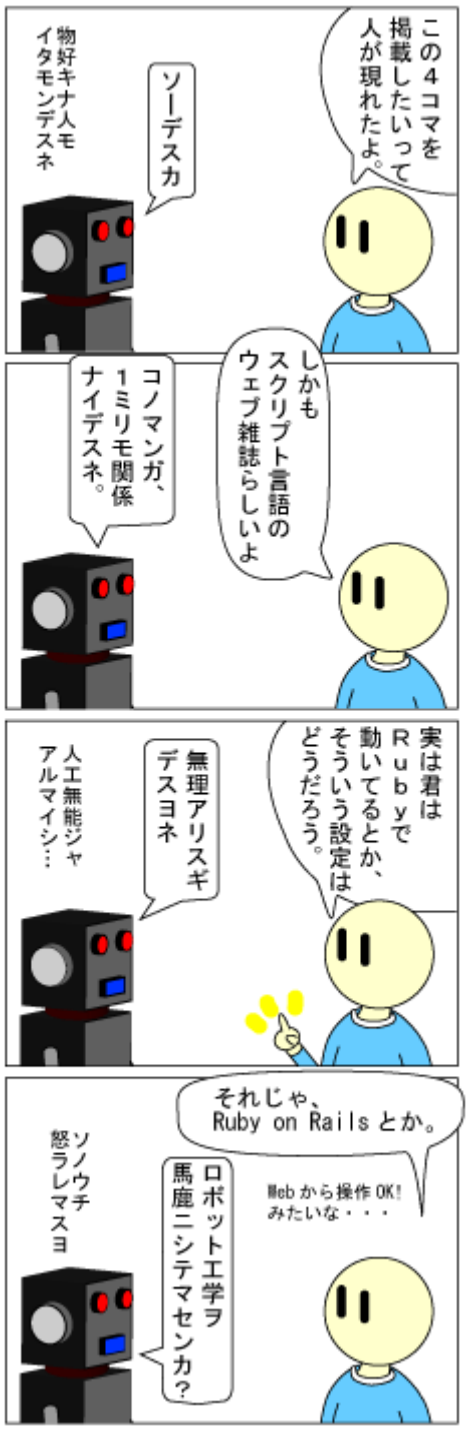
拡散ロボ粒子砲

よしおさんとロボ太

海鳥

その 37

「ロボ OS」



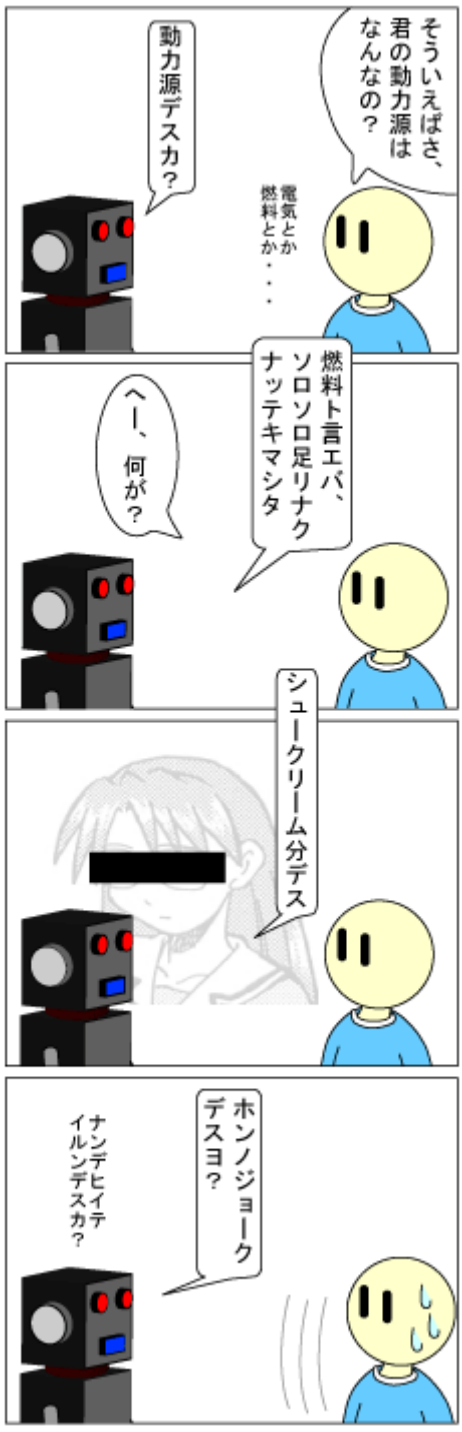
そーいや考えたことなかったわ。

よしおさんとロボ太

海鳥

その 38

「ロボ動力」



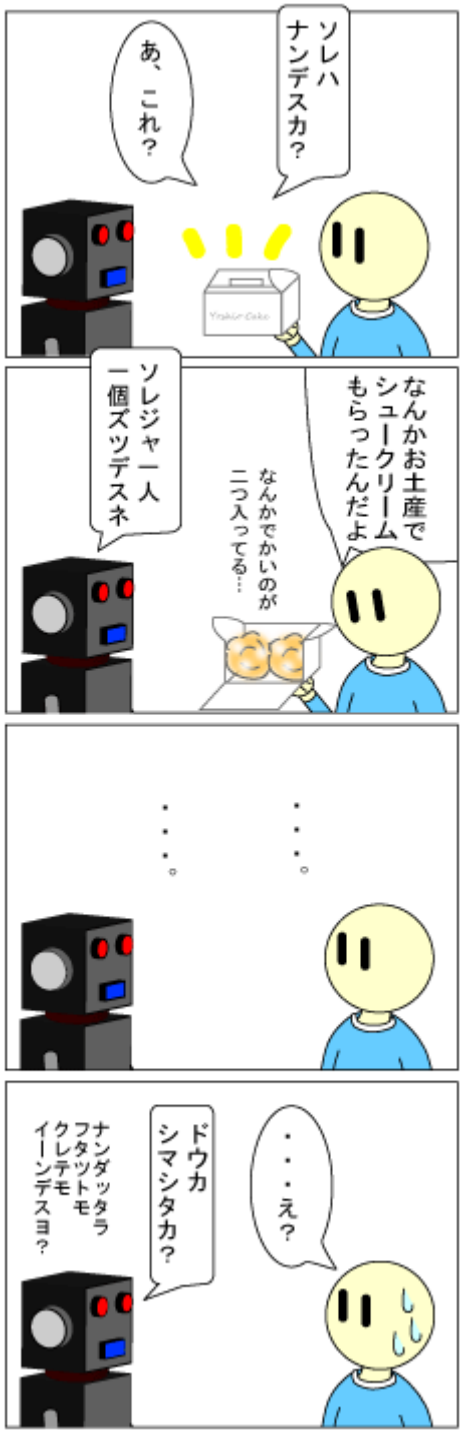
勉強とかすると減ります。

よしおさんとロボ太

海鳥

その 39

「続・ロボ動力」



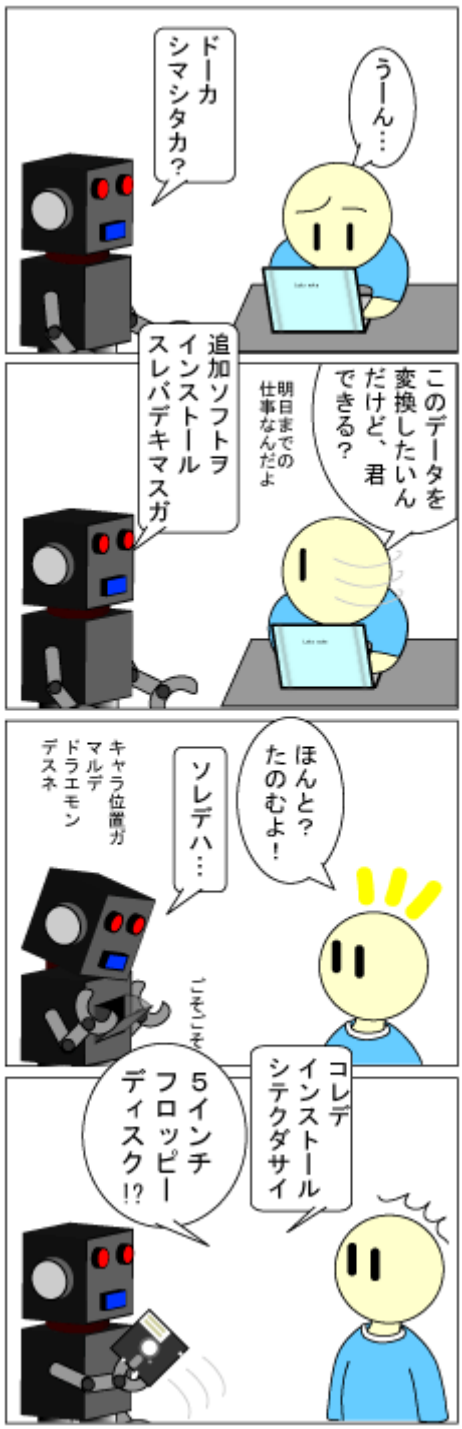
甘党ロボ

よしおさんとロボ太

海鳥

その 40

「インストール」



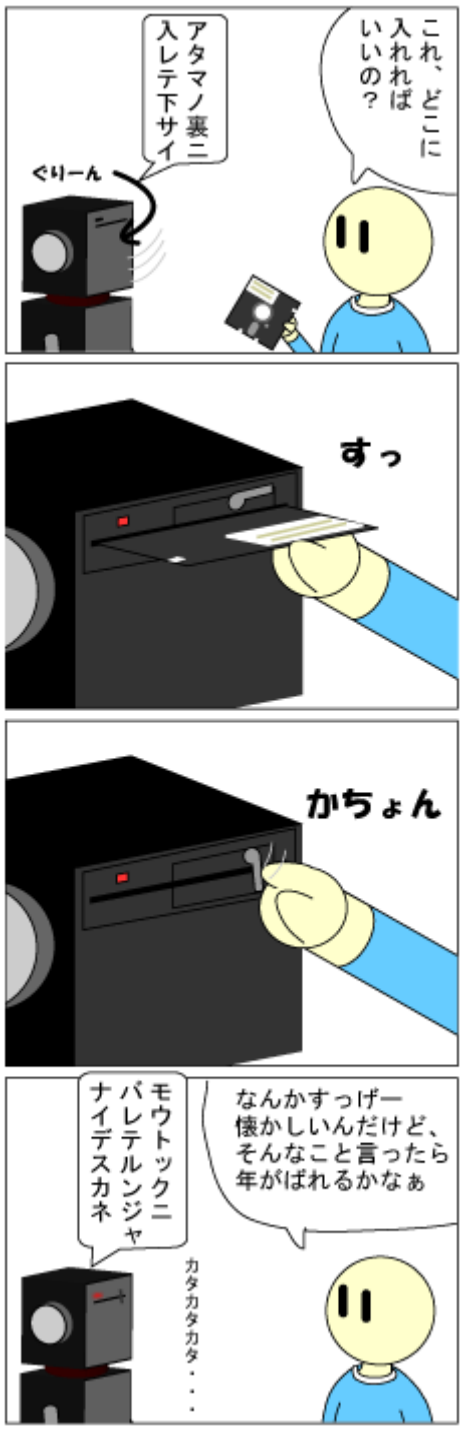
さすがに 2HD

よしおさんとロボ太

海鳥

その 41

「再インストール」



A ドライブ

よしおさんとロボ太

海鳥

その 42

「再々インストール」

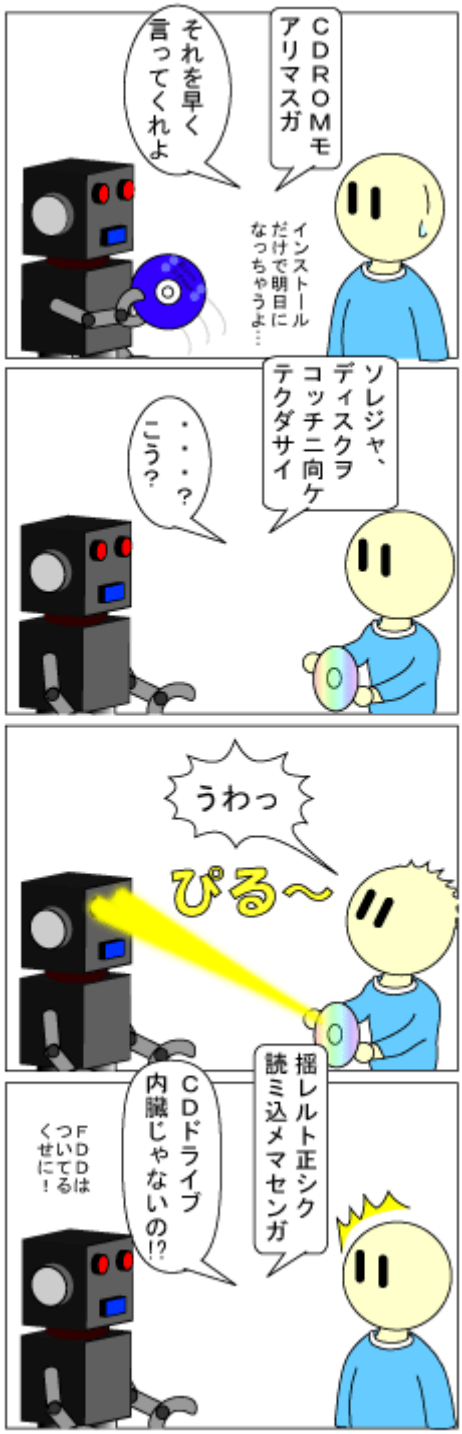
昔はいつもこんなでしたが←だから年がばれるって

よしおさんとロボ太

海鳥

その 43

「再々々インストール」



外付け CDROM ドライブ （全編掲載：2011 年 11 月 20 日）

スクリプトでパズルを可視化する

― Ruby と Graphviz で数独 ―

海鳥

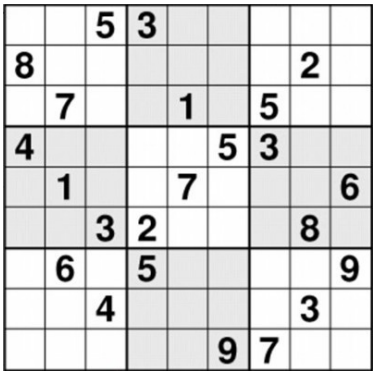
1. 概要

Graphviz ライブラリを使って、数独の回答パターンを可視化する。

2. はじめに

プログラムを趣味でやっている人なら、一度はパズルを解くプログラムを作ったことがあろうかと思う。僕もその例に漏れず、迷路の自動生成や、迷路を解くプログラムその他を作った。当時の愛機は PC-9801F（←年がバレる）、その後 PC-9801RX になったが、当時の CPU はとても非力で、適当なコードを書くととても遅かった。イラストロジックというパズルをご存知だろうか。お絵描きロジック、ピクロスとも呼ばれ、縦と横の数字をヒントにマスを塗りつぶして行くと絵がうかびあがるパズルである。いまでも人気で、携帯ゲーム機やスマホのアプリにもなっているようだ。これを自動で解いたら面白いだろうと思い、さっそくイラストロジックを解くプログラムを書いたが、最初に書いたバージョンがものすごく遅く、パズルの本に「易しい」と分類されている問題を解くのに数時間かかったりした。そこでビット演算を使って書き直したところ、イラストロジックはビット演算と相性の良い問題だったこともあって「最高難度」の問題が数秒で解けたりして感動したのを覚えている。大学のサーバに「箱入り娘」を解くプログラムを走らせたが、暴走させてメモリを食いつぶしてサーバをダウンさせたこともある。若気の至りである。

それから時は流れ、いま目の前にあるノートパソコンは、当時のスパコンなみの計算能力をもっている。本当に技術の進歩は激しいと実感する。そんなおり、職場に「世界で一番難しい数独パズル」と題された問題が貼られていた(GIGAZINE 数学のエキスパートが 3 ヶ月かけて作成した「世界一難しい数独」
http://gigazine.net/news/20100822_hardest_sudoku/)。



これは僕への挑戦である。せっかく早いマシンがあるので、スクリプトでなんも工夫しない力押しのプログラムを書いて、世界最高難度のパズルに挑んでみよう。

3. 数独の解き方

ご存知の方も多いと思われるが、数独とは９×９のマスに１から９の数字を埋めて行くパズルである。ルールは縦、横、３×３のブロックに同じ数字が入ってはいけない、というもの。すぐに思いつく解法は、それぞれのマスに１から９の候補を入れておき、他の縦、横、ブロックに入っている数字を消して行き、１つしか候補が残らなければそこの数字が確定、というもの。これで解けてしまうのが、いわゆる「易しい」問題である。難しい問題は、それだけでは答えが決まらず、候補をしぼりきれなかったところを適当に決めて進んでみて、矛盾が生じたらバックトラックして別の数字をためす、ということをする必要がある。この再帰レベルが深いほど難しい問題、ということになる。

これを高速に解こうとしたり、省メモリで解こうとしたりするとわりと大変なのだが、我々には(昔に比べれば)恐ろしく速く、無尽蔵ともいえるメモリを持つマシンがある。ここは富豪的プログラミングでいこう。つまり、コンストラクタでマスの状態を渡されたら決定論的に決まるところを埋めて、全部埋まったら完了、矛盾が生じたらエラーを返し、途中までしか埋まらなければ一番候補の数が少ないところを軸に再帰するような「数独ソルバークラス」を作れば良い。繰り返しになるが、高速化を考えなければちっとも難しくない。これを実現するスクリプトは以下の通り(sudoku.rb)。

```
#!/usr/bin/ruby

$L = 9
$L2 = $L*$L

class Array
   def sum
      self.inject(0){|result, i| result + i }
   end
end

class Grid
  def initialize (l,a)
    @level = l
    @changed = false
    @data = Array.new(a)
    @flag = Array.new($L2){|i|
      Array.new($L){|j| 1}
    }
    killall
```

```
    puts "Recursive Level: #{@level}"
    determine
  end

  def rest
    return @data.select{|i| i== 0}.size
  end

  def solved
    return rest == 0
  end

  def showflag
    i = 0
    @flag.each{|f|
      x = i % $L
      y = (i/$L)
      i = i + 1
    }
  end

  def pos2index(x,y)
    return y*$L + x
  end

  def kill_xline(i,n)
    i = (i / $L)*$L
    for j in 0..$L-1
      @flag[i+j][n-1] = 0
    end
  end

  def kill_yline(i,n)
    i = (i % $L)
    for j in 0..$L-1
      @flag[i+j*$L][n-1] = 0
    end
  end

  def kill_block(i,n)
    x = (i % $L)
    y = (i / $L)
```

```
    x = (x / 3) * 3
    y = (y / 3) * 3
    i = pos2index(x,y)
    for j in 0..2
      @flag[i+j*$L][n-1] = 0
      @flag[i+j*$L+1][n-1] = 0
      @flag[i+j*$L+2][n-1] = 0
    end
  end

  def killflag(i,n)
    @flag[i].fill(0)
    kill_xline(i,n)
    kill_yline(i,n)
    kill_block(i,n)
  end

  def killall
    @data.size.times{|i|
      killflag(i,@data[i]) if @data[i] != 0
    }
  end

  def place_number(i,n)
    return if @data[i] != 0
    x = (i % $L) +1
    y = i/$L + 1
    @data[i] = n + 1
    @changed = true
    killflag(i,n+1)
  end

  def check(num)
    t = Array.new($L){|i| 0}
    num.each{|i|
      $L.times{|n|
        t[n] = t[n] + @flag[i][n]
      }
    }
    while n = t.index(1)
      num.each{|i|
        if @flag[i][n] == 1
```

```
          place_number(i,n)
        end
        t[n] = 0
      }
    end
  end

  def check_line
    $L.times{|y|
      check(Array.new($L){|i| y*$L + i})
    }
    $L.times{|x|
      check(Array.new($L){|i| i*$L + x})
    }
    $L.times{|n|
      x = (n%3)*3
      y = (n/3)*3
      check(Array.new($L){|i| (i%3 + x) + (i/3 + y)*$L})
    }
  end

  def determine
    while 1
      @changed = false
      check_line
      return if @changed == false
    end
  end

  def solve_recursive(i,n)
    d = @data
    d[i] = n
    g = Grid.new(@level+1,d)
    return g.solve
  end

  def solve

    if solved
      puts "----------------------------"
      puts "Solved! at Level #{@level}"
      show
```

```
      exit 1
      return 1
    end

    min_index = 0
    min_value = 10
    @data.size.times{|i|
      if @data[i] == 0
        s = @flag[i].sum
        if min_value > s
          min_index = i
          min_value = s
        end
      end
    }
    if min_value == 0
      return 0
    end
    t = @flag[min_index]
    c = Array.new
    while n = t.index(1)
      c.push n+1
      t[n] = 0
    end
    x = min_index % $L +1
    y = min_index / $L + 1
    ans = 0
    c.each{|n|
      ans = ans + solve_recursive(min_index,n)
    }
    return ans
  end

  def show
      puts "----------------------------"
    $L.times{|i|
      j = $L*i
      k = j + $L-1
      puts @data[j..k].join(",")
    }
      puts "----------------------------"
  end
```

```
end

a = Array.new
while line=gets
  line.chomp.split(/,/).each{|i|
    a.push i.to_i
  }
end
g = Grid.new(0,a)
ans = g.solve

puts "This quenstion has #{ans} answer[s]."
```

実際にデータを与えて解いてみよう。データは以下のようなCSV形式で与える。

```
0,0,6,0,0,0,0,0,1
0,7,0,0,6,0,0,5,0
8,0,0,1,0,3,2,0,0
0,0,5,0,4,0,8,0,0
0,4,0,7,0,2,0,9,0
0,0,8,0,1,0,7,0,0
0,0,1,2,0,5,0,0,3
0,6,0,0,7,0,0,8,0
2,0,0,0,0,0,4,0,0
```

これをsample1.datとして保存し、引数にファイル名を与えて実行する。

```
$ ruby sudoku.rb sample1.dat
Recursive Level: 0
----------------------------
Solved! at Level 0
----------------------------
5,3,6,8,2,7,9,4,1
1,7,2,9,6,4,3,5,8
8,9,4,1,5,3,2,6,7
7,1,5,3,4,9,8,2,6
6,4,3,7,8,2,1,9,5
9,2,8,5,1,6,7,3,4
4,8,1,2,9,5,6,7,3
3,6,9,4,7,1,5,8,2
2,5,7,6,3,8,4,1,9
----------------------------
```

と、再帰レベル0、すなわち再帰せずに解けたことがわかる。
ちなみに雑誌に載っているようなもう少し難しい問題、

```
9,0,0,0,2,3,0,0,8
0,0,7,0,0,0,0,9,0
0,0,6,4,0,0,0,0,0
0,0,5,2,0,0,0,0,3
0,8,0,0,0,0,0,6,0
2,0,0,0,0,5,7,0,0
0,0,0,0,0,6,3,0,0
0,5,0,0,0,0,4,0,0
0,0,0,8,7,0,0,0,6
```

を入力として与えると、

```
$ ruby sudoku.rb sample2.dat
Recursive Level: 0
Recursive Level: 1
Recursive Level: 2
Recursive Level: 3
Recursive Level: 1
Recursive Level: 2
Recursive Level: 3
---------------------------
Solved! at Level 3
---------------------------
9,4,1,5,2,3,6,7,8
5,3,7,6,1,8,2,9,4
8,2,6,4,9,7,1,3,5
7,9,5,2,6,1,8,4,3
1,8,3,7,4,9,5,6,2
2,6,4,3,8,5,7,1,9
4,7,8,9,5,6,3,2,1
6,5,9,1,3,2,4,8,7
3,1,2,8,7,4,9,5,6
---------------------------
```

と、再帰レベル3、つまり最低でも三回再帰しないと解けないことがわかる。実際にやってみるとわかるが、人間が三回再帰しようとするとかなり手間がかかる。

それではいよいよ世界一難しい問題を解いてみよう。データと実行結果は次の通り。

```
0,0,5,3,0,0,0,0,0
8,0,0,0,0,0,0,2,0
0,7,0,0,1,0,5,0,0
4,0,0,0,0,5,3,0,0
0,1,0,0,7,0,0,0,6
0,0,3,2,0,0,0,8,0
0,6,0,5,0,0,0,0,9
0,0,4,0,0,0,0,3,0
0,0,0,0,0,9,7,0,0

$ $ ruby sudoku.rb hardest.dat
Recursive Level: 0
(中略)
Recursive Level: 7
---------------------------
Solved! at Level 7
---------------------------
1,4,5,3,2,7,6,9,8
8,3,9,6,5,4,1,2,7
6,7,2,9,1,8,5,4,3
4,9,6,1,8,5,3,7,2
2,1,8,4,7,3,9,5,6
7,5,3,2,9,6,4,8,1
3,6,7,5,4,2,8,1,9
9,8,4,7,6,1,2,3,5
5,2,1,8,3,9,7,6,4
---------------------------
```

なんと再帰レベル７である。人間がやったら数ヶ月かかると思われるが、さもありなん。しかし、それをスクリプトで、何の最適化も書けていないプログラムで実行して0.2秒で解けてしまった。

4. 解法の可視化

実際に再帰していくとき、間違ったブランチに進んでしまうと、その後全ての枝を調べても正答に至らず、何度もバックトラックする必要がある。実際にそのブランチの構造がどうなっているか可視化してみよう。RubyにはGraphvizというグラフの可視化ライブラリがある。以前にも紹介したことがあるが、GraphvizはAT&T研究所が開発したオープンソースソフトウェアで、ライブラリとして様々な言語で使うことができる。今回はRubyからGraphvizを呼び出すことにしよう。gemが使えるなら、コマンドラインから

```
$ gem install ruby-graphviz
```

とするだけでインストールされるはずだ。後はスクリプトで

```
require 'rubygems'
require 'graphviz'
```

とすれば使うことができる(gem から入れた場合は rubygems を require するのを忘れずに)。
グラフの可視化を使ったソースコードは以下の通り(sudoku_graph.rb)。

```
#!/usr/bin/ruby
require 'rubygems'
require 'graphviz'

$L = 9
$L2 = $L*$L

$graph = GraphViz::new("G", { :type => "digraph", :use => "dot", :output =>
"png", :file => "sudoku.png"})

class Array
   def sum
      self.inject(0){|result, i| result + i }
   end
end

class Grid
  def initialize (l,a)
    @level = l
    @changed = false
    @data = Array.new(a)
    @flag = Array.new($L2){|i|
      Array.new($L){|j| 1}
    }
    killall
    puts "Recursive Level: #{@level}"

    determine
    s = ""

    $L.times{|i|
      j = $L*i
      k = j + $L-1
```

```
      s = s + @data[j..k].join(" ")
      s = s + "¥n"
    }
    @node = $graph.add_node(s)
  end

  def get_node
    return @node
  end

  def rest
    return @data.select{|i| i== 0}.size
  end

  def solved
    return rest == 0
  end

  def showflag
    i = 0
    @flag.each{|f|
      x = i % $L
      y = (i/$L)
      i = i + 1
    }
  end

  def pos2index(x,y)
    return y*$L + x
  end

  def kill_xline(i,n)
    i = (i / $L)*$L
    for j in 0..$L-1
      @flag[i+j][n-1] = 0
    end
  end

  def kill_yline(i,n)
    i = (i % $L)
    for j in 0..$L-1
      @flag[i+j*$L][n-1] = 0
```

```
    end
  end

  def kill_block(i,n)
    x = (i % $L)
    y = (i / $L)
    x = (x / 3) * 3
    y = (y / 3) * 3
    i = pos2index(x,y)
    for j in 0..2
      @flag[i+j*$L][n-1] = 0
      @flag[i+j*$L+1][n-1] = 0
      @flag[i+j*$L+2][n-1] = 0
    end
  end

  def killflag(i,n)
    @flag[i].fill(0)
    kill_xline(i,n)
    kill_yline(i,n)
    kill_block(i,n)
  end

  def killall
    @data.size.times{|i|
      killflag(i,@data[i]) if @data[i] != 0
    }
  end

  def place_number(i,n)
    return if @data[i] != 0
    x = (i % $L) +1
    y = i/$L + 1
    @data[i] = n + 1
    @changed = true
    killflag(i,n+1)
  end

  def check(num)
    t = Array.new($L){|i| 0}
    num.each{|i|
      $L.times{|n|
```

```
        t[n] = t[n] + @flag[i][n]
      }
    }
    while n = t.index(1)
      num.each{|i|
        if @flag[i][n] == 1
          place_number(i,n)
        end
        t[n] = 0
      }
    end
  end

  def check_line
    $L.times{|y|
      check(Array.new($L){|i| y*$L + i})
    }
    $L.times{|x|
      check(Array.new($L){|i| i*$L + x})
    }
    $L.times{|n|
      x = (n%3)*3
      y = (n/3)*3
      check(Array.new($L){|i| (i%3 + x) + (i/3 + y)*$L})
    }
  end

  def determine
    while 1
      @changed = false
      check_line
      return if @changed == false
    end
  end

  def solve_recursive(i,n)
    d = @data
    d[i] = n
    g = Grid.new(@level+1,d)
    $graph.add_edge(@node, g.get_node)
    return g.solve
  end
```

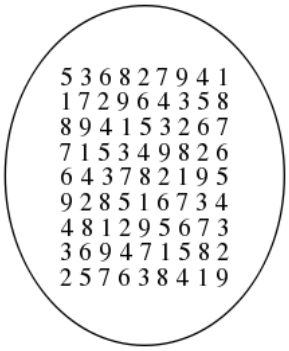
```
  def solve

    if solved
      puts "----------------------------"
      puts "Solved! at Level #{@level}"
      show
      return 1
    end

    min_index = 0
    min_value = 10
    @data.size.times{|i|
      if @data[i] == 0
        s = @flag[i].sum
        if min_value > s
          min_index = i
          min_value = s
        end
      end
    }
    if min_value == 0
      return 0
    end
    t = @flag[min_index]
    c = Array.new
    while n = t.index(1)
      c.push n+1
      t[n] = 0
    end
    x = min_index % $L +1
    y = min_index / $L + 1
    ans = 0
    c.each{|n|
      ans = ans + solve_recursive(min_index,n)
    }
    return ans
  end

  def show
      puts "----------------------------"
    $L.times{|i|
```

```
      j = $L*i
      k = j + $L-1
      puts @data[j..k].join(",")
    }
      puts "----------------------------"
  end
end

a = Array.new
while line=gets
  line.chomp.split(/,/).each{|i|
    a.push i.to_i
  }
end
g = Grid.new(0,a)
ans = g.solve

puts "This quenstion has #{ans} answer[s]."
$graph.output()
```

実際に使ってみよう。まずは再帰しない問題から。

```
$ ./sudoku_graph.rb sample1.dat
```

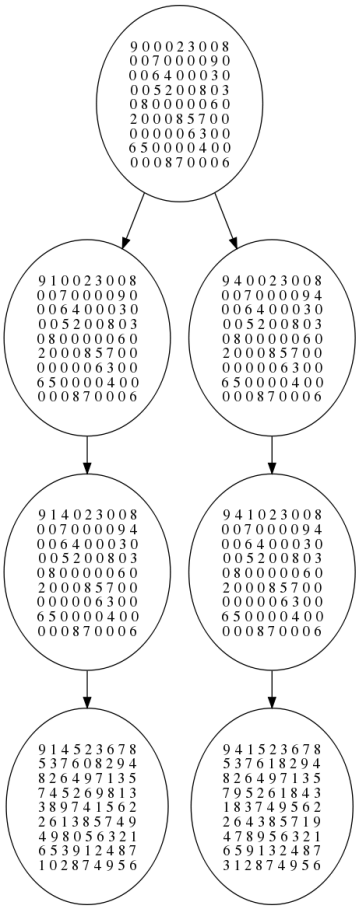
実行すると以下のような sudoku.png が出力される。



再帰なしなので、いきなり答えが出る。

次に、再帰レベル３の問題。

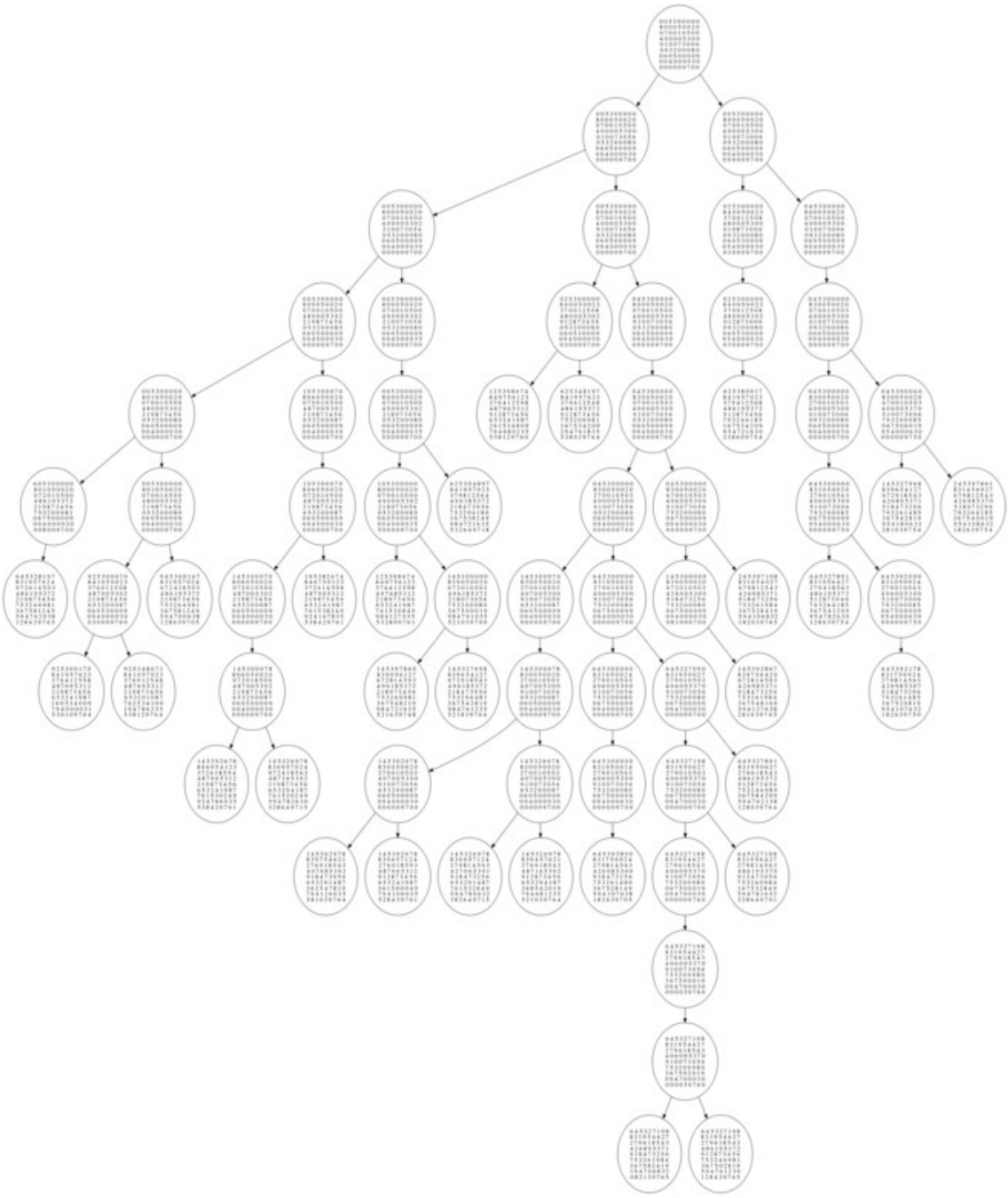
```
$ ./sudoku_graph.rb sample2.dat
```



再帰レベルは３だが、再帰の軸を間違えなければさほどブランチは増えないことがわかる。

さて、いよいよ世界一高難度の問題を食わせてみよう。

```
$ ./sudoku_graph.rb hardest.dat
```



得られた図は大きすぎて、原寸大ではとても紙面には載せられない。正答は再帰レベル 7 にあるが、誤ったブランチに行くと再帰レベル 12 までさまようことになる。なお、このグラフはどこを軸に再帰するかによって異なる。下手なマスを軸に再帰したらもっと広いグラフになるかもしれない。

5. おわりに

Ruby で数独ソルバを作り、可視化してみた。昔は論理演算などを作って高速化しないと高難度の問題は実時間では解けなかったものだが、今では再帰の度にクラスのインスタンスを作る、などという富豪的プログラムでもなんの問題もなく解けてしまう。ちなみに、数独ソルバはデバッグも含めて半日足らずで作る事ができた。プログラムが得意な人ならもっと早く組めるだろう。世界一難しい数独の問題は、人間が解くと数ヶ月かかるそうだから、プログラムする手間をかけてもそっちの方が早い、ということになる。これを「ずるい」と思う人もいるだろうが、僕の考えは違う。コンピュータはあくまで紙と鉛筆の延長にある。それを使いこなすのは人間の技術であり、それも含めて人間の能力と考える。コンピュータがチェスの名人を破り、将棋でも女流名人を破った。これをコンピュータの勝利、人間の敗北と考える人もいるが、そのプログラムを作ったのもコンピュータを開発したのも人間であり、僕は人間の勝利と考えたい。しかし今後、コンピュータはますます速く、かしこくなり、できることが増えて行くだろう。コンピュータが究極に進化した時、なお人間にしかできないものが残るかどうか・・・そう考えると、いつまでも「コンピュータの勝利」を「人間の勝利」と考えることができるか、ちょっと自信がなくなってくるが・・・。

（投稿：2011 年 11 月 14 日）

覚醒から進化へ　試論 I

jscripter

－本書は、この九年間で私がしてきた仕事を、本の形で初めて語るものである。これほど遅くなったのは、ひとえに希望と不安のせいであった。次の角を曲がればその向こうには新しい景色が見えるかもしれないという希望が、それまでの仕事に区切りをつけると言う作業をためらわせ、またこうしたためらいは、自分たちの理論はもうすぐ時代遅れになうかもしれない、という不安を増幅させるものなのだ。－
(マイケル・ポランニー著、高橋勇夫訳「暗黙知の次元」、ちくま学芸文庫、2003 年)

１．はじめに

僕は、21 世紀に入った年に Web 日記を付け始め、しばらくして、Awking Project を提案した。Automated Web Knowledge, Information and News Gathering Project の略である。訳せば、自動化知識・情報・ニュース収集プロジェクトとなろうか。 これは後に、Awakening Project、覚醒プロジェクトと称するようになる。「世の中は変わる、覚醒せよ」と言うメッセージのつもりだった。Automated Web *Awaking* Knowledge, Information and News Gathering Project の略と言うことになっているが、アナグラムに文字をどのように当てはめるのかは定かでない。

当時とは言っても 21 世紀に入ったばかりのころ、Perl の LWP:Simple と言うモジュールの get メソッドを利用すれば、URL をキーにして、Web に存在する HTML を自在に取り込み、テキストから必要な情報を抽出し、記録・加工することが、欲すれば、誰にもできるようになっていたのである。これを大規模に行えば、Google になる。まったく同じことをしても仕方がないが、新しいアイデアのもとにおいて、使いようによっては大変に役立つツールとなることに気が付いたのである。

今年は東北大震災が起き、福島第一原発事故の終息も依然として定かではない。最近では、TPP 参加問題が浮上しているが、何を議論すべきなのかさえ、見通せない感じがしている。IT ビジネスの世界では、リベラル・アーツとテクノロジーの境界領域で仕事をするのを好んだスティーブ・ジョブズが亡くなり、大きな衝撃を与えた。ギリシャ・イタリアの経済危機により信任を失ったパパンドレウ・ベルルスコーニ両首相の辞任劇。中米新旧二大国の動向は、世界の行方は、・・・未来は情報の大海の中で混沌としている。

筆者は、Awakening Project を「デスクトップと Web を融合する」システムとして捉え、CGI の枠組のなかでインターネット時代の知的生産の技術として発展させようと試みてきた。そして 10 年が経過した。ここで、現在の新しい状況と自分の関心を織り込みつつ、次のステップを構想するのも無駄ではあるまいと思われる。

プログラマにとって、自分のアイデアがいつ陳腐化してしまうか、他のプログラムによって何時無駄なものに成り果てるのか、は悪夢のようなものであるが。

２．「メモる」システム

Awakening Project の初期の実験の後、RSS リーダーを含む統合的なシステムを 2004 年に発表した。jperl を使った CGI で実現している。「メモる」システムを作り始めた頃は、まだPerl 5.8.0 が出たばかりで、use encoding 'shiftjis'などを実用的に使えるレベルではなかったからだ。また、UTF-8 を自在に使えるようになるまでにはさらに年月を要したのである。

RSS リーダーとメモカレンダーを中核とした「メモる」システムは、現在ではCodez-ine(翔泳社)に発表したAtom フィードのリーダーを加え、新しいデータは、UTF-8 で取り扱うシステムに置き換えている。

「メモる」システムは元々はメモカレンダーの日付にリンクしてメモを付けるシステムだったが、RSS やAtom の記事についてもメモを書けるように拡張できることに気が付いた。これはあらゆるデータやコンテンツにメモなり注をつけることができることを意味した。ただメモを日付単位で取り扱う仕様に問題があった。異質なデータが一つのメモに混在することになるからである。あとで述べる更新日記も同様な課題を抱えている。

３．Wemo をつくる

2008 年の Wemo は 、一つの日付に複数のメモを対応させるように、データモデルを変更しようとする試みであった。が、ロードマップを 2009 年に作成したまま、途中で中断している。基本的なデータモデルの変更は完了しているが、データ形式について、 まだアイデアを練りたいと言うことも頭を掠めているからである。UTF-8 の文字コードを使用している。

４．更新日記作成システム

著者の更新日記作成システムと並行して、非公開情報をデスクトップに蓄積するためのシステムとして「メモる」システムは作られてきている。「メモる」システムはCGI であるのに対して、更新日記作成システムはCGI とともにエディタ上で動作するシステムでもある。文字コードは最初からシフト JIS のままで運用している。

前述の「 メモる」システムやWemo がさらに発展する機会を得ることができないのは、実は、HTML で書かれている更新日記そのものがメモになっているからである。ハイパーテキストによる情報処理はそれだけで本質的に強力である。あらゆるものをハイパーテキスト日記に埋め込むことが可能なのである。

５．Web の進化と CGI

Web はHTML やXML/RDF などのテキストからなる。テキスト処理のプログラミングにおいては、万能のスイスアーミーナイフであるPerl が適していることは言うまでもない。

Perl は、コマンドラインや CGI だけでなく、標準入出力を編集に使えるエディタの Zed、TeraPad、Dana などでも動作する。そのなかでも、ハードウェアの進化と HTML5 や CSS3 の登場によって Web の表現力が飛躍的に増大した現在、CGI の重要性はさらに高まっている。実際、Google Docs や Gmail などを見れば、ブラウザがデスクトップに近い性能を実現しつつあるのである。

しかしながら、Web の表現力の増大を見つめつつも、それを裏側で支える JavaScript を学ぶにはコストが高いと、鬱陶しく感じている面もある。ビジュアルな GUI の操作性は情報処理においてそれほど本質的なものではないのではないか、と言う疑問が HTML5 に踏み込むのを妨げている。もっとシンプルで十分なのでは？

６．世界を記述するもの

－ 暗黙的認識をことごとく排除して、すべての知識を形式化しようとしても、そんな試みは自滅するしかないことを、私は証明できると思う。 －
(マイケル・ポランニー「暗黙知の次元」、前掲、原著:1966 年)

ポランニーの暗黙的認識とは、等価かどうかは別にするにしても、チョムスキーの言う「言語使用の創造的な側面」のことを言っていることになるわけで、デカルトが主張したように機械的な概念では説明できない(ノーム・チョムスキー「言語と知識 －マナグア講義録 言語学編－」、産業図書、1989 年、原著:1988 年;「デカルト派言語学」、みすず書房、1976 年、原著:1966 年)。

Web は自然言語と人工言語(HTML/XML/RDF/Javascript/CSS)のハイブリッドから成り立っている。画像や音声データは人工言語の記述のなかに URL として埋め込まれている。Web の人工言語の部分は機械的な概念で説明でき、ブラウザが読み取って再生することができる。すなわち、機械的な処理をすることが可能である。問題は自然言語の部分であり、人が読まないと意味が生じない。

しかしながら、人間の生得的能力である暗黙的認識を、機械にも理解できるように記述する方法の試みが、XML/RDF である。すなわち、Semantic Web である。Machine-readable な Web が Semantic Web なのである。

結局は、人間が、機械に理解できるように付加的な人工言語の記述をすることになる。一種の埋込型のプログラミングである。データのパターンや辞書を利用して、プログラムによって機械的にマークアップすることもできる。そのプログラムは人間が書くわけだが。
すべての暗黙的認識を形式化できないとしても、可能な範囲で形式化する努力は続いている。

僕が今注目しているのは、XTM と DITA である。

7. XTM と DITA

XTM は XML Topic Maps の略である。以前、Jack Park, Sam Hunting 編「XML Topic Maps Web のためのトピックマップ開発」(星雲社、2004 年、原著:2003 年)を購入していたのだが、情報を汎用的に表す最重要な概念と感じて時折眺めながらも、何年も積読状態にしていた。最近、DITA から連想して、またもやひっくり返している。

XTM は 2002 年に ISO 標準(ISO/IEC 13250:2002)となっているが、広く知られているとは言い難い。トピックマップで情報を表す要素として、トピック(topic、主題となる任意の概念を表す)、連想(association、トピック間の関係を表す)、事象(occurrences、トピックとトピックに関わるリソース間の関係を表す)がある。このように説明を書き出すと、ものごとはそう簡単ではないとすぐ感じてしまう。連想や事象の中にもトピックは存在しているからだ。世界には多層・多重・循環の構造が内在していて、それは暗黙知のなかに解消されて取り扱われている。XTM では、必要に応じて、あるストーリーの中で部分的に明示化していくことになる。個人のレベルでは暗黙的にデータを取り扱えるので、データとしての共有を必要としない限り、また自らデータとしての再利用を必要としない限り、XTM の積極的な利用は躊躇してしまう。

しかしながら、Perl には、TM と言うモジュールがあり、Linear Topic Map notation(LTM)や Compact Topic Maps(CTM)、AsTMa などの記述を XTM に変換可能であり、利用に一考の余地がある。

DITA は Darwin Information Typing Architecture の略である。「ダーウィン進化的情報型定義アーキテクチャ」とでも訳すとよいかもしれない。DITA は XTM とは直接関係ないが、DITA は「トピック」と「マップ」が基本要素として定義されている XML である。マップはコンテンツ要素であるトピックを集めて文書を定義する。

DITA の特徴は、特定の組織の文書を表現するために特殊化された新しいトピック型、エレメント型、属性を定義することを可能にしていることだ。特殊化されたトピック型として、概念(Concept)、タスク(Task)、参照情報(Reference)、用語集(Glossary) を標準で提供している。マップにおいては書籍構成要素について、特殊化されたマップとして書籍マップ(BookMap)を定義している。継承(inheritance)を使って特殊化(specialization)しているとしているが、詳細は確認する必要がある。最近では、DITA を ePub に変換することも試みられている。

DITA のマップは XTM のトピックの連想や事象に対応するものと考えられ、概念的にはトピックマップから多くを受け継いでいるものと言えるが、サンプルを見ると、HTML に近く表現を強く意識したものになっている。たとえば、

http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/topic.html

から、topic 要素のサンプルを引用しておこう。

```
<topic id="topic">
 <title>Some little topic</title>
 <body>
  <p>Here's a <b><i>cute</i></b>,
  <b>little</b> topic.</p>
  <ul>
   <li>Some item</li>
   <li>Another item</li>
  </ul>
 </body>
</topic>
```

DITA は XML というより HTML そのものに近く、親しみやすいと言えるかもしれない。

８．最後に - HTML からの進化

意味的に暗黙的な HTML から明示的な XML に進化すると、Web は Machine-readable になる。Web の自動化においては必要なことではあるが、個人的には必ずしも必要ではない。さて、柔軟なパターンと辞書を使うか、固定的なマークアップを使うか、考えどころではある。

次のステップでは具体的な応用について書ければと考えている。

９．参照文献

Awakening project
http://homepage1.nifty.com/kazuf/awking.html

Wemo
http://text.world.coocan.jp/TSNET/?plugin=related&page=Wemo

更新日記 日曜プログラマのひとりごと
http://homepage1.nifty.com/kazuf/renewal.html

Topic Maps — XML Syntax
http://www.isotopicmaps.org/sam/sam-xtm/

The Linear Topic Map Notation
http://www.ontopia.net/download/ltm.html

ISO/IEC 13250-6: 2010, Information technology — Topic Maps — Part 6: Compact Syntax
http://www.isotopicmaps.org/ctm/ctm.html

AsTMa [AsTMa Topic Map Processing]
http://astma.it.bond.edu.au/

tm - search.cpan.org
http://search.cpan.org/dist/TM/bin/tm

DITA version 1.1, Language Specification, OASIS Standard, 1 August 2007
http://docs.oasis-open.org/dita/v1.1/OS/langspec/ditaref-type.html

(投稿: 2011 年 11 月 20 日)

編集後記

jscripter

世界の混迷は深まるばかりだが、コンピュータやネットワークの発展は止まらない。もしかしたら、コンピュータやネットワークが生み出し続ける大量の情報が世界の無秩序さの増大をもたらしているのかもしれない。

と、思いつきをまず書き留める。それはともかく、世界は新興国を含めた経済の発展と人口の増大と共に資源は足りなくなりつつある。「覚醒と進化」の最初に書いたことに加えて、アフリカには飢饉で飢えている人々がおり、中東の政治情勢の行方もよく見えない。中東には民主主義は根付きにくい。民主主義とは世界の観点から見れば少数派なのだということらしい。

世界の状況を認識し、何をどうしたらよいのかを考えることができるようにすることは知識情報処理の目的であらねばならいないだろうと思う。日本国内の混迷を見聞きするにつけ、国内の問題でさえ、自分はどこまで理解できているのだろうと思わざるをえない。

残念ながらお疲れ気味なので、ここらへんで筆を置こう。最近は、新聞のスクラップを始めたりしている。アナログと手作業に戻れと・・・

以上。

(投稿: 2011 年 11 月 20 日)

ＴＳＮＥＴスクリプト通信

ISSN 1884-2798　出版地: 広島市

2011 年 11 月 21 日　4.2.002 版（GAawk 関連スクリプト収録、フォント統一）
2011 年 11 月 20 日　4.2.001 版

投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと

編集委員会(投稿順)

Ｙさ　saw[at]s7[dot]wh[dot]qit[dot]ne[dot]jp
ムムリク　qublilabo[at]gmail[dot]com
海鳥　kaityo256[at]gmail[dot]com
jscripter　jscripter9[at]gmail[dot]com

著作権

使用許諾・配布条件

免責事項

編集ソフトウェア

OpenOffice.org 3.2.1 Writer

発行所

一次配布所: TSNET スクリプト通信刊行リスト
http://text.world.coocan.jp/TSNET/?TSNET%E3%82%B9%E3%82%AF%E3%83%AA%E3%83%97%E3%83%88%E9%80%9A%E4%BF%A1%E5%88%8A%E8%A1%8C%E3%83%AA%E3%82%B9%E3%83%88

TSNETスクリプト通信 第4巻第2号(通算第14号)
発行: TSC編集委員会 発行日: 2011年11月20日
ISSN: 1884-2798 出版地: 広島市 創刊: 2008年5月7日