ARTICLE

再帰的思考のすすめ

中川正樹
東京農工大学

今，教員生活を終わろうとしているこの時点で，これまでに「美しい」と思った再帰的な考え方をまとめてみたいと思う．難しいと思われるところは読み飛ばしていただきたい．

数学的帰納法

今から約 50 年前，高校生のときに数学で一番美しいと思ったのは数学的帰納法であった．n=1 のときに成り立つことと，n=k–1（あるいは n<=k–1）のときに成り立つと仮定し，n=k でも成り立つことを示して，したがって，すべての自然数 n で成り立つという証明法である．例として，次の等式を証明してみよう．

$$1+2+3+4+\cdots+n=\frac{1}{2}n(n+1)$$

n=1 のときは，$1=\frac{1}{2}1(1+1)$ なので，明らかに成り立つ．次に，n=k–1 のときに成り立つと仮定する．つまり，n に k–1 を代入して，

$$1+2+3+4+\cdots+(k-1)=\frac{1}{2}k(k-1)$$

が成り立つと仮定とする．これを用いて，次のように n=k でも成り立つことを示す．

$$\begin{aligned}&1+2+3+4+\cdots+k\\&=1+2+3+4+\cdots+(k-1)+k\\&=\frac{1}{2}k(k-1)+k=\frac{1}{2}k(k+1)\end{aligned}$$

したがって，n=1 で成り立って，2 で成り立って，. . . ，k–1 で成り立って，k で成り立つというやり方である．

数式微分

大学 3 年生のときに，LISP という言語で数式微分のプログラムを書いた．数値微分ではなく，入力に x^2 を与えると $2x$ を返す微分である．変数がアルファベット 1 文字だけで，数も 1 桁，かつ，四則演算だけの数式は次のように定義できる．下で，「｜」は「または」の意味である．

式＝項＋式｜項－式,
項＝因子×項｜因子／項,
因子＝(式)｜変数｜数,
変数＝ a|b|…|x，数 =0|1|…|9.

ある数式の x による微分 D は次に従う．

D(項＋式) = D(項) + D(式),
D(項－式) = D(項) − D(式),
D(因子×項) = D(因子) ×項＋因子× D(項),
D(因子／項)
　= (D(因子) ×項－因子× D(項)) ／ (項)^2,
D((式)) = (D(式)),
D(変数) = if 変数 = x, then 1 else 0,
D(数) = 0.

数式微分のプログラムは，表記の違いはあるが，原理的にほとんどこの通りである．そして，実行は，たとえば次のようになる．

D(x^2+3x+1)=D(x^2)+D(3x+1)
=D(x)×x+x×D(x)+D(3x)+D(1)
=1×x+x×1+D(3)×x+3×D(x)+0

$=x+x+0\times x+3\times 1+0=2x+0+3+0$

$=2x+3$

プログラムを書いてみると，数式の微分そのものよりも，結果を整理する（たとえば，$1\times x=x$，$x\times 1=x$，$x+x=2x$，$0+3+0=3$，無駄な括弧を外すなどのための）プログラムの方がはるかに長くなったのを記憶している．詳しく学びたい読者は文献 1)，2) を参照されたい．

典型的な再帰プログラム

再帰のプログラムで典型的なのは「ハノイの塔」の解法手順を出力するプログラムである．ベトナムのハノイにある寺院には，3 本の柱（A, B, C）と，真ん中に穴の開いた大きさがすべて異なる 100 枚の石でできた円盤があるという．元々はすべての円盤が柱 A に刺さっていて，僧侶たちが 1 枚ずつ移動し，すべての円盤を柱 C に移動しようとしている．僧侶が 100 枚の円盤を移動し終わった瞬間，この世は消滅するという．図 -1 は円盤が 5 枚のときの課題を示す．円盤は重ねられて柱 A に刺さっており，それらをすべて柱 C に移動する．ただし，1 枚ずつしか移動できず，しかも，小さな円盤の上にそれよりも大きな円盤を重ねてはならない．どの円盤も移動後は，いずれかの柱に刺しておかなければならない．そのために柱 B を補助として利用する．

さて，n 枚の円盤を柱 A から柱 C へ，柱 B を補助に使って移動する手順を示す言語 C のプログラム 1 を完成してみよう．1 枚のときは自明である．n 枚のときは，上の n–1 枚を柱 B に移し，一番下の円盤を柱 C に移してから，柱 B の n–1 枚を柱 A による補助で柱 C に移せばよい．プログラムの□の中には，a, b, c いずれかの変数が入るので，適切なもので埋めて実行してみよう．

円盤 1 枚を 1 分で動かせるとして，移動を始めてからどれぐらいの時間で終了するか考えてみてほしい．安心して眠れると思う（なかなか計算できなくて眠れないかもしれないが……）．ちなみに，ハノイには 10 回ぐらい行ったが，いまだにこの寺院には行き当らない．

コンパイラ

筆者が若いころは計算機科学がまだ黎明期で，日本ではまだその教育が確立しておらず，大学院に入ってから，英国の Essex 大学の修士課程に留学する機会を得た．そこでは，初期の人工知能の論文もたくさん読んだが，学部科目で，言語 C などの高水準言語から機械語に翻訳するコンパイラというプログラムを読んだ．プログラムが上例の数式と同

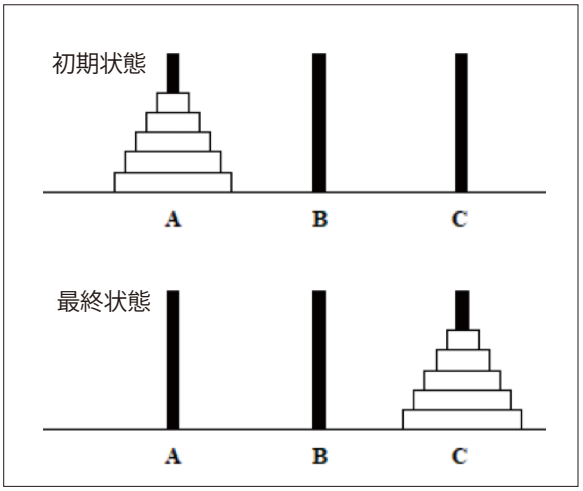


図 -1　ハノイの塔（5 枚のとき）

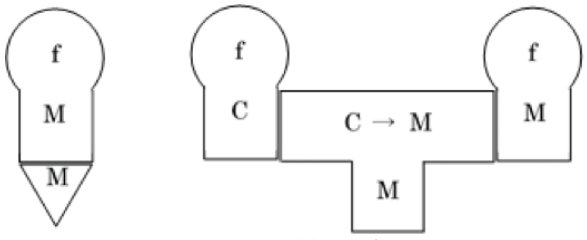
```
#include <stdio.h>
void hanoi(char a, char b, char c, int n)
/* move n disks from a to c using b */
{
   if(n==1) printf("Move a disk from %c
      to %c.¥n", a, c);
   else {
      hanoi(□, □, □, n - 1);
      printf("Move a disk from %c to %c.
      ¥n", □, □);
      hanoi(□, □, □, n - 1);
    }
}
int main()
{
   int n;
   printf("Tower of Hanoi¥n");
   printf("How many disks? ");
   scanf("%d", &n);
   hanoi('A', 'B', 'C', n);
}
```
プログラム 1　解法手順を表示するプログラム

じように厳密な再帰的文法で定義されることを利用して，高水準言語から機械語への変換を再帰法に従って行う．そのプログラムの行数は予想よりはるかに少なかった．この分野の成果は計算機科学の金字塔の１つと言えるだろう．

そして，小さい言語からより大きい言語に拡大していく方法（bootstrapping）にも感嘆した．図 -2（a）に示すように，機械語 M で動作するコンピュータを▽で中に M，機械語 M で記述された機能 f を実行するプログラムを鍵穴形で下に M，上に f で書く．プログラムを実行させることで，f の機能が発揮する．

コンパイラは機能として，言語 C から機械語 M に変換するので，図 -2（b）で示すように，鍵穴の上の半円を四角に変形し，中に C → M と書く．言語 C で書かれた任意の機能 f のプログラムを同じ機能を持つ機械語 M のプログラムに変換する．

最初のコンパイラは小さい言語（μ C と書こう）のために作り，その言語 μ C でフルスペックの言語 C のコンパイラを書けば（図 -3 の左端），そのコンパイラをコンパイルすることで，フルスペック C のコンパイラが得られる（図 -3 中央）．これを動作させれば，任意の機能 f のプログラムをコンパイルできる（図 -3 右）．言語 C だけでなく，別の言語でもよい．興味のある人は Earley [3] を読んでほしい．



(a) 機能 f の実行　　(b) コンパイラの機能

図 -2　記述言語と機能の実行

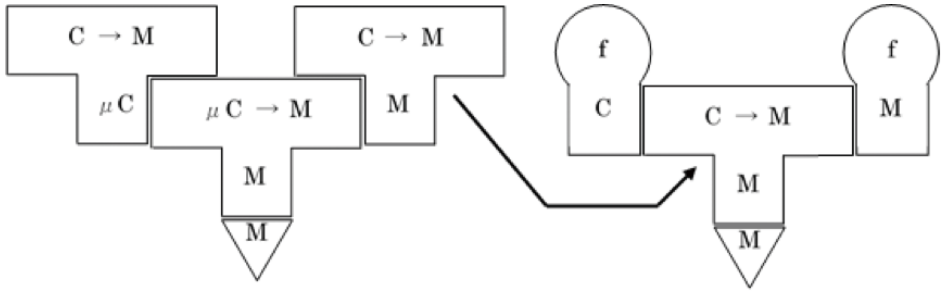


図 -3　小さい言語のコンパイラを作って大きい言語のコンパイラを作る

分割統治法

自然数は１から始まって，それに１を加えたものも自然数という一番単純な構造を持つ．木構造は幹から次々に枝に分かれてどこかで葉になる構造を持つ（そうでない変な植物もあるが）．プログラムは枝や葉に種類がある木構造である．こうした構造を処理するプログラムを書くとき，構造を分解して部分ごとに解法を適用し，それらの結果をまとめることで構造全体を処理する方法を分割統治法と呼ぶ．要素数が n のときに，たとえば，分割しないと n^2 の処理時間になる方法を，上手に分割することで $n \log n$ で処理できるようになる．数学やプログラムあるいはデータのように明確な構造を持つ対象だけでなく，さまざまな対象で分割統治法は採用されている．学校のクラス分けやグループ分け，企業や組織の部や課への分割とそれらの管理である．大規模な問題を効率的に処理できるプログラムを書ける人は大組織の管理や運営にも能力を発揮するはずである．したがって，誰もがプログラムを書いてみることに賛成である．逆に，プログラムを書いたこともない経営者や管理者が，情報システムの企画責任者になることにはリスクを感じざるを得ない．メガ銀行が統合するときに起きたトラブル，大手コンビニの電子決済システムの穴など，枚挙にいとまがない．

落語

修士課程を出て，経済的な理由から博士課程に行かずに農工大の助手として採用してもらった．講座の教授は高橋延匡（のぶまさ）先生だった．計算機システムで多くの業績を残された先生なのだが，落語好きで「えんきょうさん」と呼ばれていた．その先生から落語には再帰の概念も登場するという話を聞いた．それは「あたまやま」である．

あるケチな男がサクランボの種を捨てるのが惜しいと飲み込んだら，頭のてっぺんから桜が生えてきて，春になると満開になる．町人が花見に押しかけて宴会をする．しかし，そのうるささに耐えかねて，桜を抜いてしまう．すると，その穴に雨が溜まって池になり，魚が増える．今度は，町人が昼夜を構わずに釣りに来る．うるさくて眠ることもできない．その男は「いっそ死んでしまえ」と，その池に身を投げる．無理矢理に図にすると図 -4 のようになる．

別の例は「のっぺらぼう」である．ある商人の男が身投げをしようとする若い女を助けてみると，顔に目も鼻も口もないのっぺらぼうである．男は「世の中，なまじ目や鼻がついているために苦労している女は五万といる」となだめつつも，内心では血が引く思いで，蕎麦屋の屋台に駆け込む．息も切れ切れにその話をすると，蕎麦屋は「こんな顔ですかい」とのっぺらぼうの顔で振り向く．男は慌てて家に逃げ帰り，奥さんにその話をする．すると，奥さんが「こんな顔？」と振り向く．男は気を失う．しばらくして，奥さんに起こされて男は目を覚ます．「変な夢でも見てたんじゃない」と言われて，夢だったのかと，その夢の話をする．そうすると奥さんが「こんな顔？」と振り向く．男は気を失う，…….

これらの話は繰り返し（loop）ではない．再帰（recursion）なのだ．

図 -4　あたまやま

振り返って

再帰が美しいのは，一番単純な場合の処理を定義し，単純でない場合には問題を分解して部分の処理に自分自身を呼んで結果をまとめる記述をすればよいことである．何回ループするなどの記述は不要である．先人は再帰プログラムを実行できるようにするために，計算を途中で保留して，次の計算を始め（その計算をまた途中で保留し，次の計算を始める，……），結果が返ってきたときに計算を再開するためのスタックというデータ構造を生み出した．

当方は農工大の助手に採用してもらって 41 年間，手書きパターン認識とタブレットのユーザインタフェースの研究をし，実用化もできた．

この原稿とあまり関係のない研究のように思われるかもしれないが，課題解決のさまざまな局面で再帰が登場した．まさに，自分の思考法の柱の 1 つが再帰であった．

参考文献

1) 平野拓一：記号数式処理，http://www.takuichi.net/hobby/symbolic/
2) 渡邉慶一：LIST でやる記号微分，https://www.slideshare.net/KeiichiWatanabe/lisp-102975121
3) Earley, J. and Sturgis, H.: A Formalism for Translator Interactions, Communications of the ACM, 13, 10, pp.607-617 (1970).

（2020 年 1 月 12 日受付）

中川正樹（正会員） nakagawa@cc.tuat.ac.jp

1979 年東京農工大学工学部助手．1997 年教授．現在，副学長．手書きパターン認識とユーザインタフェースなどの研究・教育に従事．理学博士．2016 年文部科学大臣表彰，東京都功労者表彰など受賞．