

3.3 コラッツの問題

3に縁が深いのは円周率だけではない。次の規則で作られる数の問題を考えてみよう。

はじめに勝手な自然数を用意する。次の項は「前の項が偶数なら2で割り、奇数なら3倍して1を足す」というものだ。たとえば50から始めると

50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, ...

のような数列が出来上がる。数列はどこまでも続くように思える。しかし、そうではないのだ。52から先を計算してほしい。さして労力をかけずに様子が分かるはずだ。様子を先に言ってしまうと、どうも数列は..., 4, 2, 1, ...のように、結局1に戻るように思える。

この数列はコラッツ¹の問題と呼ばれる未解決問題である。未解決というのは、あらゆる自然数はこの規則で必ず1になるのか、という点だ。例外は知られていない。**Haskell**が解決してくれるわけではないが、この数列の行方を調べてみよう。それには偶数と奇数の判断ができなくてはならないが、その方法は学習済みだ。

しかし、今度は何かの値をひとつ求めるのではなく、数列を表示したい。それには前に目にしたリストを使うとよいだろう。

[hs script]

```
collatz :: Integer -> [Integer]
collatz 1 = [1]
collatz n
  | even n = n : collatz (n `div` 2)
  | odd  n = n : collatz (3 * n + 1)
```

スクリプトはお馴染みの再帰を用いて数列を生成していることが分かるだろうか。でも、その前に `Integer -> [Integer]` がいままでとは違うね。`[Integer]` は整数値のリストということである。すなわち、`Integer -> [Integer]` は、整数型の引数を取り整数型のリストを返す関数という意味である。

さて、スクリプトの説明の前に実際の動作を示す方が理解の助けになるだろう。最初に示した例である50から始めてみよう。

(ghci env.)

```
*Main> collatz 50
[50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
```

やっぱり1にたどり着いた。では、スクリプトを説明しておこう。

¹ローター・コラッツ (1910–1990) : ドイツの数学者。

再帰であることをほのめかすのは2行目の `collatz 1 = [1]` である。`collatz 1` となるところが基点で、それは `[1]` というリストを返す。それはどういうときに起こるかという、たとえば `collatz 2` のときである。なぜなら、ガードの記述において、`n` が偶数なら `n` を2で割った値を引数にして `collatz` 関数が実行されることが分かるからだ。

そして `n = 2` のときは、具体的に `2 : collatz 1` が実行される。`n :` という記述はリストの先頭に `n` を追加する。`collatz 1` は `[1]` であったから、`2 : collatz 1` は `[2,1]` を返すことになる。

理解を深めるために、今度は `n = 50` から先の処理を追ってみよう。この場合は、まず `collatz 50` から始まり、50は偶数だからガードの偶数における分岐 `50 : collatz 25` となる。`collatz 25` は25が奇数のため、ガードの奇数における分岐に従って `25 : collatz 76` となる。これは、始めの `50 :` と合わさって、結局 `50 : 25 : collatz 76` となる。すると、あとはこの繰り返しだから

$$50 : 25 : 76 : 38 : 19 : \dots \dots : [1] \rightarrow [50,25,76,38,19, \dots \dots, 1]$$

が出来上がるのである。

そこで、前回 `pivalue' 715` で求めた円周率1001桁の値を `collatz` 関数に食わせたらどうなるだろう。それには、`collatz (pivalue' 715)` を実行すればよい。少々時間がかかるけれど、なんと最後は1になるのだ。こんな例を見れば、もうどんな自然数だってオッケーって思えるが、この程度の値は無限に大きな値から見ればカスみたいなもんだ。コラッツの問題が未解決であるのも領(うなず)けるのではないだろうか。

ところで、コラッツの問題には様々な亜種がある。それは「前の項が偶数なら2で割り、奇数なら3倍して1を足す」という規則を変えることだ。たとえば「前の項が偶数なら2で割り、奇数なら5倍して3を引く」みたいにすることである。これはすぐに試すことができる。

[hs script]

```
collatz' :: Integer -> [Integer]
collatz' 1 = [1]
collatz' n
  | even n = n : collatz' (n `div` 2)
  | odd  n = n : collatz' (5 * n - 3)
```

最初のように50を与えてみよう。

(ghci env.)

```
*Main> collatz' 50
[50,25,122,61,302,151,752,376,188,94,47,232,116,58,29,142,71,352,176,88,44,
,22,11,52,26,13,62,31,152,76,38,19,92,46,23,112,56,28,14,7,32,16,8,4,2,1]
```

ふむ、ふむ。いい感じだ。では `collatz' 45` はどうだろう、って調子で試すとんでもないこ

とになる (ctrl+c で止まるのだったね)。この場合は数が巨大になる一方で、いつ終わりがくるのか予想もできない。いろいろな規則を調べるのは簡単だから、ぜひやってみよう。余力があれば偶数と奇数で分岐するだけでなく、3で割って1余るときの規則、2余るときの規則、割り切れるときの規則、というように3通りの分岐にしてもよいだろう。しかし、なかなかコラッツの規則のようにうまくいかないことが分かるだろう。3倍して1を足す規則は、妙に均衡がとれているのである。

コラッツの問題とは系統が異なるが、ライフゲームと呼ばれるシミュレーションがある。これは、方眼に区切られた平面に置いたマスについて、次の規則で新たに石を置いたり取り除いたりするものである。

	A	B	C	D	E
1					
2			●	●	
3		●	●		
4			●		
5					

- 石があるマスの周囲8マスについて、周囲に石が1個以下、もしくは石が4個以上のときはマスの石は取り除かれる。たとえば3Cのマスの石は、周りに4個の石があるので取り除かれる対象である。
- 石があるマスの周囲8マスについて、周囲に石がちょうど2個、もしくは石がちょうど3個のときはマスの石はそのまま残る。たとえば4Cのマスの石は、周りにちょうど2個の石があるので残る対象である。
- 石がないマスの周囲8マスについて、周囲に石がちょうど3個のときはマスに新たに石が置かれる。たとえば4Bの空いたマスは、周りにちょうど3個の石があるので新たに石を置く対象である。

規則は以上の3種類だけである。ただし、石を取り除いたり新たに置いたりするのは、すべての石と空きマス进行调查したとき同時に行うものとする。さて上の図において、1Aから5Eまでの25マスについて規則を当てはめてみよう。取り除く対象の石は“◎”に、残る対象の石は“●”のままに、新たに石を置く対象のマスは“・”にすると

	A	B	C	D	E
1					
2		●	●	●	
3		●	◎		
4		・	●		
5					

⇒

	A	B	C	D	E
1					
2		●	●	●	
3		●			
4		●	●		
5					

のように変化するのである。そして、次の配置においても同じ規則を当てはめていくと、コラッツの問題のように次々と配置が変化していく。

ライフゲームがコラッツの問題と異なるのは、最後の状態がある特定の配置に落ち着くのではないことだ。あるときは平面に石がひとつもなくなって終了するかもしれない。また、あるときは同じ配置が繰り返されるようになるかもしれない。場合によっては、際限なく石が増えることもある。

ライフゲームがコラッツの問題と似ているのは、規則を変えれば様々な亜種ができることだ。そして、周囲に石がちょうど3個のときにマスに新たに石が置かれる規則が、絶妙な規則になっているところが同じなのだ。

いま、この段階で **Haskell** によるライフゲームを作るのは無理だけれど、ライフゲームはコンピュータ向きのシミュレーションである。自分でプログラムを組むことに挑戦するのもよいし、インターネットでライフゲームを探してもよいだろう。