4.4 偶数の分解と素因数分解

それではゴールドバッハの予想を確認してみよう。これまでのスクリプトの蓄積で簡単に書けるだろう。

(ghci env.)

```
*Main> let goldbach n = [(m, n-m) | m <- [3, 5..n], isprime m && isprime (
n-m), m < n-m]
*Main> goldbach 88
[(5,83),(17,71),(29,59),(41,47)]
*Main> goldbach 124
[(11,113),(17,107),(23,101),(41,83),(53,71)]
```

これも 1 行のスクリプトで出来てしまった。画面では最初の`*Main>` 以下が 2 行に分かれているが、実際は 1 行で書くべきものである。注意しよう。

数 n を 2 個の素数和で表すとき、$n = p + q$（p, q は素数）のようにするのは効率が悪い。ひとつを m とすれば、もうひとつは $(n - m)$ でよい。素数のペアは集合の表記に通じるリストに出力している。ペアなのでタプル表記にしてあるのは、双子素数を求めたときと同じだ。

調べるのは n と $(n - m)$ が共に素数であるかどうかなので、以前作った `isprime` 関数に引数で与えている。`A && B` の表記は、`A` と `B` が共に真であるときに真を返すので、3 以上の奇数から選んだ n を順次調べればよい。これだけでよいのだが、`m < n-m` を加えておくと、$(5,\ 83)$、$(83,\ 5)$ のような同じ組を排除できる。

ゴールドバッハの予想の土いじりから少し離れたところを掘り返すことになるけれど、せっかく素数に馴染んできたところなので、**素因数分解**でもいじってみたい。素因数分解とは、ある数を素数の積で表すことである。具体例は $100 = 2 \cdot 2 \cdot 5 \cdot 5$ や $365 = 5 \cdot 73$ などである。当然、素数は素因数分解できない。

素因数分解をするには候補となる数を次々と素数で割っていき、割ることができた素数の積で構成すればよい。たいていのプログラミングでは、

`p_0 = 2,  p_1 = 3,  p_2 = 5,  p_3 = 7,  ...`

のような素数の一覧表を用意して、候補となる数を `p_i` で順次割ることになる。

このように素数表を用意すれば、余計な除数で割り算をする無駄が省けて、大変効率的である。しかし、素数表をどこまで準備しておくかが重要な問題である。素数のリストなら以前作った `primes` 関数が使えるのだが、この関数は素数のリストを作るために相応の計算をしていたので、無駄を省くことにはならない。そこで、ここでは相変わらず効率が悪い割り算をするが、素数表をその場で

2

作成しながら素因数分解するスクリプトを書いてみた。

[hs script]

```
intfactor :: [Int] -> [Int]
intfactor (1:xs) = xs
intfactor (x:xs) =
    let divprime = last [a | a <- primes' x, x `mod` a == 0]
        quotient = x `div` divprime
    in  intfactor (quotient : divprime : xs)
```

基本的には見なれたコードだろうが、let〜in という構文が新しい。

ここでの素因数分解の方法は、たとえば 60 を例にとれば、まず最小の素数である 2 で割れるかを試すだろう。60 は 2 で割れて 30×2 となる。次に 30 も同様に 2 で割れて $15 \times 2 \times 2$ となる。今度は 15 の番だが、これは 2 で割れないので、次の素数 3 で割り算を試す。すると割れることが分かり $5 \times 3 \times 2 \times 2$ となって、素因数分解の完成である。

人が素因数分解をする場合はこれでよいが、**Haskell** では最後の商が 5 になったとき、何を根拠に計算をやめればよいだろうか。人は、最後の 5 は素数という理由で計算をやめるだろう。もちろん **Haskell** もそうしてよいのだが、ここではもう一度、商である 5 を素数で割ることにしている。そして $1 \times 5 \times 3 \times 2 \times 2$ までの分解をし、先頭が 1 であることを理由に計算をやめることにした。

つまり、[1,...] というリストになったときが素因数分解の完成と判断する。したがって、ここが基底部分となるから、(1:xs) の xs 部分に素因数分解の因数が列挙されているはずである。

さて、関数はリストの head である x と tail である xs のうち、x が素数で割れるかどうか調べればよい。そのために primes' x で素数リストを作成し、そこから取り出した a が x で割れるかを調べ、割れた a だけの集合を作る。primes' 関数は小さい素数がリストの最後から並ぶようになっていたはずだから、last 関数で因数を divprime へ代入する。商は x `div` divprime となる。

これを、商、因数、既存のリストの順につなげれば、分解が一段進んだ状態になるので、さらにこれを intfactor 関数へ与えればよい。ここでも再帰である。

このことを繰り返していくと、最後に商が 1 になるので、このとき intfactor (1:xs) がマッチして素因数分解が完成するのである。

(ghci env.)

```
*Main> intfactor [720]
[5,3,3,2,2,2,2]
*Main> intfactor [9991]
[103,97]
```

しかし、結果がリスト表示になるという理由で、引数に与える数までリストの形を強要されるのは我慢ならないと思う。そんなときは

(ghci env.)
```
*Main> let intfactor' n = intfactor [n]
```

と定義し直せばよい。すると

(ghci env.)
```
*Main> intfactor' 720
[5,3,3,2,2,2,2]
```

のようにできる。さらに表示を、たとえば 720 = 5*3*3*2*2*2*2 みたいにしたいときは別の道具が必要になる。この道具は庭いじりには過ぎたものだが、あとで目にできるだろう。$720 = 5 \cdot 3^2 \cdot 2^4$ のようにしたいときも別の道具が必要だ。

ただ $5 \cdot 3^2 \cdot 2^4$ を、5 が 1 個、3 が 2 個、2 が 3 個、のようなリストにすることなら直ちにできる。それには Data.List **モジュール**をインポートする。そのためのコマンドは import だ。

(ghci env.)
```
*Main> import Data.List
*Main Data.List>
```

するとプロンプトが*Main Data.List> に変わるので、モジュールが使えるようになったことが分かる。これで group 関数が使えて

(ghci env.)
```
*Main Data.List> group (intfactor' 720)
[[5],[3,3],[2,2,2,2]]
```

のように因数ごとにグループ分けされる。group を用いて正しくグループ分けするためには、リストがソートされている必要があるのだが、intfactor 関数は小さい因数から調べているので、リストは自然にソートされた状態になっているので心配ない。

ただ、これではある因数がとてもたくさん出てきたら数えるのに苦労するだろう。でも、**Haskell** がリストに含まれる要素を数えれば済む話だ。それは length 関数がやってくれる。また、リストの中の各要素—それはリストの形をしている—に length を適用するのだから、map を使うのが適切だろう。

(ghci env.)
```
*Main Data.List> map (\xs -> (head xs, length xs)) [[5],[3,3],[2,2,2,2]]
[(5,1),(3,2),(2,4)]
```

しかし単に length を適用するだけでは、“何が”、“何個” あるか分からなくなってしまうね。だからその前に (何が, 何個) かを示す関数を定義しておこう。\xs -> (head xs, length xs) がそれである。リスト化された要素の数値は皆同じなので、head xs で “何が” を知ることができる。

xs の前に\が付いているのは、この場限りの **λ 関数**（ラムダ関数）を定義することを意味している。この臨時関数を map を用いてリストの要素（つまりリスト）に適用すればよいのだ。ここでは直接 [[5],[3,3],[2,2,2,2]] を与えているが、このリストは intfactor 関数が出力したものだから、実際は次のように書けばよいだろう。

(ghci env.)

```
*Main Data.List> map (\xs -> (head xs, length xs)) $ group $ intfactor' 720
[(5,1),(3,2),(2,4)]
```

ちゃんと、$720 = 5^1 \cdot 3^2 \cdot 2^4$ であることが分かるようになったね。でも “$” って何？

まず map は、map f [リスト] という使い方をするのだったね。この場合、その定義に従えば map (\xs -> (head xs, length xs)) (group (intfactor' 720)) と書けばよい。でも、ちょと ( ) が目障りかな。そのようなときは$を用いて ( ) を減らすことができる。“$” を “末尾に) がある (” と見てみよう。慣れれば ( ) より見やすいはずだ。