8 $1+1$（その 2）

さあ、だいぶ長い道を旅して、ようやく自然数が定義できた。いよいよ $1+1$ を考えるときだ。自然数を定義したとき、“a の後者” という表現に出会ったね。これから $1+1$ を考えるのに、その言葉遣いは冗長すぎる。これからは a の後者を $suc(a)$ と表現しよう[*1]。

まず、1 が出てきた。1 が何だったかというと、“0 の後者” として定義された数である。たったいま導入した表現を用いれば $suc(0)=1$ である。ここで 0 の性質—0 に (何か) を足しても (何か) でしかない—が生きてくる。足し算 $+$ がどういう行為か知らないが、0 の性質の前では $0+1=1$、すなわち $0+1=suc(0)$ なのである。0 に対して $+1$ することが $0+1=suc(0)$ と書けるなら、1 に対して $+1$ することは $1+1=suc(1)$ と書けてよいだろう。つまり、n に対して $+1$ することが n の後者と定義できるだろう。

□ **自然数の定義と足し算の定義**

- ペアノの公理によって構成したものを自然数とする
- $suc(n)$ は n の後者である（このことから $suc(0)=1$）
- $n+1=suc(n)$

この定義で $1+1=2$ だろうか。ペアノの公理から構成したのは $\emptyset$, $\{\emptyset\}$, $\{\{\emptyset\},\emptyset\}$, $\{\{\{\emptyset\},\emptyset\},\{\emptyset\},\emptyset\}$, $\{\{\{\{\emptyset\},\emptyset\},\{\emptyset\},\emptyset\},\{\{\emptyset\},\emptyset\},\{\emptyset\},\emptyset\}$, ... だったので、これらに順に数詞 0, 1, 2, 3, 4, ... を与えて自然数としたんだね。数詞の列を眺めると 1 の後ろに 2 があるので、$suc(1)=2$ であることはすぐ分かる。ところで、$n+1=suc(n)$ と定義したので $1+1=suc(1)$、すなわち 2 である。以上。

あれ？　ここは今回の旅の最大の景勝地であるはずなのに、意外にあっさりと通過してしまうんだ。せっかくの景勝地なのに、期待はずれに終わった気分だろう。でも、安心してほしい。この景勝地には裏があるのだ。

裏を探る前に確認をしよう。$suc(0)=1$, $suc(1)=2$, $suc(2)=3$, ... であることと $n+1=suc(n)$ を合わせれば、$1+1=suc(1)=2$, $2+1=suc(2)=3$, $3+1=suc(3)=4$, ... であることは容易に分かる。その結果、$1+1=2$, $2+1=3$, $3+1=4$, ... が見える。そして、これをうまく使うと $m+n$ が自在に作れる。$2+3$ なら次のようになるだろう。

$$\begin{aligned}
2+3 &= 2+(2+1) && [\ 3 \to suc(2) \to 2+1\ ] \\
&= 2+(1+1+1) && [\ 2 \to suc(1) \to 1+1\ ] \\
&= (2+1)+1+1 && [\ \text{結合律}\ ] \\
&= 3+1+1 && [\ 2+1 \to suc(2) \to 3\ ]
\end{aligned}$$

[*1] a の後者 := successor of a

$$= 4 + 1 \qquad [\ 3 + 1 \to suc(3) \to 4\ ]$$
$$= 5\ . \qquad [\ 4 + 1 \to suc(4) \to 5\ ]$$

少々冗長すぎる嫌いはあるけれど、ちゃんと計算ができている。何のことはない。自然数に $suc(n)$ を定義できれば、$1 + 1$ はもとより任意の足し算が自在にできる。このことを **PowerShell** で実現してみよう。

[ps script]
```
PS C:\Users\Yours > function suc($n) {
>> if ($n -eq 0) {return 1}
>> (suc($n-1))+1 }
>>
PS C:\Users\Yours > function add($n, $m) {
>> $n + (suc($m-1)) }
>>
PS C:\Users\Yours > add 2 3
5
```

最初の関数 `suc($n)` は、$suc(0) = 1$, $suc(n) = n + 1$ の定義である。次の関数 `add($n, $m)` は $n + m = n + suc(m - 1)$ を計算する。スクリプトは少々いんちき臭いけれど、正しい答をきっちり出してくれる。いんちき臭い部分は、add 関数にある。$n + m$ は suc 関数によって $n + 1 + 1 + \cdots + 1$ に展開されるが、add 関数は $n + 1 = suc(n)$ であることを無視して、単に $n + 1 + 1 + \cdots + 1$ を計算するだけだからだ。本当に定義に忠実なスクリプトにするのはとても大変なので、許してもらいたい。

しかし残念なことに、この足し算の定義は不完全である。試しに `add 2 0` として $2 + 0$ をやらせてみよう。エラーが返ってくるはずだ。なぜかと言えば、足し算の定義に $n + 0$ がないからだ。スクリプトがエラーを起こすのは、定義に問題があるというより、`suc($n-1)` の引数が 0 にならずオーバーフローを起こすためである。$0 + n$ ならそのようなことは起こらないので、足し算の定義に**交換律** $a + b = b + a$ を認めよう。すると add 関数には、交換律に相当するスクリプトが加わり、add 関数は以下のようになる。

[ps script]
```
PS C:\Users\Yours > function add($n, $m) {
>> if ($m -eq 0) {$n, $m = $m, $n}
>> $n + (suc($m-1)) }
>>
PS C:\Users\Yours > add 2 0
2
```

今度はエラーにならず、きちんと答がでる。これで自然数に足し算を完全に定義できた、と思ってはいけない。実は、これだけでは $0 + 0$ が計算できないのだ。

ペアノの公理では 0 はいかなる数の後者でもないはずだった。つまり、1 や 2 を $suc(0)$ や $suc(1)$ と表すこ

とができても、0 だけは suc(何々) と表せないことになる。このことは、1 から後の数には先の定義が意味を持つが、0 については意味を持たないということだ。これは唯一の例外である。

別に $0+0$ が定義できなくてもいいじゃん、という考えもあろう。数学では、0 で割ってはいけないという約束があるので、その約束のもとでは $0\div0$ のみならず $2\div0$ やら $3.14\div0$ やら、無数の計算できない事例が生ずる。それに比べれば、あとひとつ $0+0$ も計算しない約束を取り付けることぐらい...ってわけにはいかないだろう。だって、いままで $0+0=0$ って計算してたんだから。