『離散構造』 6 章 (帰納) の演習問題の解答

問題 1 (集合の帰納的定義)

(a) a, b, c の 3 文字からなる文字列で、同じ文字が 3 回以上連続して現れるものの集合 A を帰納的に定義せよ。たとえば、$abc \notin A$ であるが、$aabbbbbac \in A$ である。

(答) いろいろな解答があり得るが、一番簡単なのは、「同じ文字が 3 回連続した文字列 (aaa など) の左右に任意の文字列をつないだもの」として定義することであろう。つまり、以下の通り。

- $aaa \in A$.
- $bbb \in A$.
- $ccc \in A$.
- $s \in A \Rightarrow as \in A$.
- $s \in A \Rightarrow bs \in A$.
- $s \in A \Rightarrow cs \in A$.
- $s \in A \Rightarrow sa \in A$.
- $s \in A \Rightarrow sb \in A$.
- $s \in A \Rightarrow sc \in A$.

(b) (10 進法の) 自然数をあらわす文字列の集合 B を定義せよ。たとえば、$123 \in B$ であるが、$00123 \notin B$ である。(自然数を普通に書くとき、頭に 0 を書いてその後に数字を続けることはないので。)

(答) この問題の場合は、先頭の文字が 0 でなければよく、2 番目以降の文字には何の制限もない。そこで、以下のように定義できるような気がする。(ただし、 $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ と置いた。)

- $x \in \Sigma - \{0\} \Rightarrow x \in B$.
- $(x \in \Sigma \wedge s \in B) \Rightarrow sx \in B$.

。。。と答えを出したら、ちょっとまずい。というのは、これでは 0 が含まれないからである。大学 (および大学以降) の数学では、0 も立派な自然数である。しかし、上記の帰納的定義に 0 を加えると、おかしくなる。たとえば、次のものは、まずい定義である。

- $0 \in B$.
- $x \in \Sigma - \{0\} \Rightarrow x \in B$.
- $(x \in \Sigma \wedge s \in B) \Rightarrow sx \in B$.

これでは、$012 \in B$ となってしまう。どうやら、この方式の帰納的定義では、うまく B を 1 回で定義できないようである。そこで方針を変えて、まず、「0 以外の自然数をあらわす文字列の集合 B_0」を定義して、それに 0 という要素を加えよう。

- $x \in \Sigma - \{0\} \Rightarrow x \in B_0$.
- $(x \in \Sigma \wedge s \in B_0) \Rightarrow sx \in B_0$.

と帰納的定義される B_0 を使って、求める集合 B は、$B = B_0 \cup \{0\}$ と定義できる。

(注) 「なんだ、これでは、B は帰納的に定義されていないではないか」と怒った人がいるだろう。しかし問題文をよく見てほしい。前問と違い、本問は、「帰納的に定義せよ」とは書いていない。帰納的定義を利用して解答する問題ではあるが、B そのものは、(B_0 を帰納的に定義した後では) 帰納的定義を使わないのである。なんだか、ひっかけ問題のようで若干気がとがめるが、世の中にはこんなこともある。

(c) 前問と同様に、(10 進法の) 小数をあらわす文字列の集合 C を定義せよ。たとえば、$123 \in C$ や $123.123 \in C$ であるが、$.123 \notin C$ であるし、$123.500 \notin C$ である。(ここでは、123.500 のように末尾に 0 が余計についている文字列も排除して考える。)

(答) この問題は、やや複雑であるので、順番に考えていこう。

まず、「小数」という言葉が、「整数」を含むことに注意しよう。つまり、123 のような整数も C の要素である。言いかえれば、前問の解答の B を使うと、$B \subset C$ である。

次に、「純粋な小数」(整数でない小数、小数点以下に数字が 1 文字以上ある小数) をあらわす文字列を考えよう。このような小数の整数部分は、当然ながら、B の要素である。そのあとに小数点 (ピリオド) が来て、さらに小数部分が来る。小数部分は、B_0 とはちょうど逆で、「0 で終わらない任意の文字列」であるので、以下のように定義できる。

- $x \in \Sigma - \{0\} \Rightarrow x \in C_0$.
- $(x \in \Sigma \wedge s \in C_0) \Rightarrow xs \in C_0$.

B_0 とは逆に、0 でない数字からはじめて、前の方に数字を追加していく形で、C_0 を定義している。よって、「純粋な小数」の集合は、

$$\{s.t \mid s \in B \wedge t \in C_0\}$$

と定義でき、最終的に、「小数 (整数あるいは純粋な小数)」をあらわす文字列の集合 C は、以下のように定義できる。

$$C = B \cup \{s.t \mid s \in B \wedge t \in C_0\}$$

問題 2 (関数の帰納的定義) 以下の関数を帰納的に定義せよ。

(a) 自然数のリストが与えられた時、その最大値を返す関数 max。ただし、引数が空リストのときは 0 を返すとする。

答. リストに対する関数なので、空リストの場合と、そうでない場合に場合分けするのがよい。この問題はさらに、リストの先頭要素が、残りのリストの最大値より大きいかどうかで場合分けする必要がある。

$$\mathtt{max}(L) = \begin{cases} 0 & \text{if } L = \langle\rangle \\ x & \text{if } L = \mathrm{cons}(x, L') \text{ and } x \geq \mathtt{max}(L') \\ \mathtt{max}(L') & \text{if } L = \mathrm{cons}(x, L') \text{ and } x < \mathtt{max}(L') \end{cases}$$

テキストや例題の関数と違う点としては、右辺の if の中で、関数 max を $\mathtt{max}(L')$ として使っている点である。これは若干気持わるいが、そこでの引数 L' は、L より「小さい」リストなので、問題ない。

(b) 自然数のリストが与えられた時、それに含まれる数の積を返す関数 mult. たとえば、$\mathtt{mult}(\langle 2, 5, 3\rangle) = 30$ となる。なお、$\mathtt{mult}(\langle\rangle) = 1$ とする。

(答) これは、実は前問よりもっと簡単である。

$$\mathtt{mult}(L) = \begin{cases} 1 & \text{if } L = \langle\rangle \\ x * \mathtt{mult}(L') & \text{if } L = \mathrm{cons}(x, L') \end{cases}$$

これで「離散構造」の解答としては満点であるが、プログラミングの課題であれば、もっと工夫したいところである。たとえば、引数となるリストの中に、1 つでも 0 要素があれば、残りの数のかけ計算を

せずに、いきなり 0 という答えを返して計算を終了する方が効率が良い。したがって、以下のようにしたくなるところである。

$$\texttt{mult}(L) = \begin{cases} 1 & \text{if } L = \langle\rangle \\ 0 & \text{if } L = \text{cons}(0, L') \\ x * \texttt{mult}(L') & \text{if } L = \text{cons}(x, L') \wedge x \neq 0 \end{cases}$$

これでも同じ関数を定義していることは容易にわかるであろう。

(c) 自然数のリストのリストが与えられた時、その要素のリストたちを全部連結した、1 つのリストを返す関数 `flatten`. たとえば、$\texttt{flatten}(\langle\langle 1,2,3\rangle, \langle 4,5\rangle, \langle 6,7,8,9\rangle\rangle) = \langle 1,2,3,4,5,6,7,8,9\rangle$ となる。

(答) この問題を解くためには、2 つのリストを連結する関数 `append` を使う。これは

$$\texttt{append}(\langle 1,2,3\rangle, \langle 4,5\rangle) = \langle 1,2,3,4,5\rangle$$

といった計算をする関数であり、その定義はテキストに掲載されている。また、$\texttt{append}(L_1, L_2)$ を $L_1 \mathbin{@} L_2$ と書く。

この関数を使うと、`flatten` 関数は驚くほど簡単に定義できる。

$$\texttt{flatten}(L) = \begin{cases} \langle\rangle & \text{if } L = \langle\rangle \\ x \mathbin{@} \texttt{flatten}(L') & \text{if } L = \text{cons}(x, L') \end{cases}$$

この定義に納得いかない人は、$\texttt{flatten}(\langle\langle 1,2,3\rangle, \langle 4,5\rangle, \langle 6,7,8,9\rangle\rangle)$ などを、手動で計算してみるとよい。

問題 3 (自然数に対する帰納法) 関数 $f : \mathcal{N} \to \mathcal{N}$ が以下の条件を満たすとする.

$$\begin{cases} f(0) = 1 \\ f(n) \leq 2f(n-1) + 2^n - 1 \quad (n > 0 \text{ の時}) \end{cases}$$

このとき，任意の自然数 n に対して，

$$f(n) \leq n \cdot 2^n + 1$$

であることを n に関する帰納法で証明せよ.

答. 以下の不等式が任意の自然数 n に対して成立することを n に関する帰納法で証明する。

$$f(n) \leq n \cdot 2^n + 1$$

(base case) $n = 0$ の時に成立するかどうか見ると、不等式の左辺は $f(0) = 1$ であり、右辺は 1 であるので成立している。(base case 終了)

(step) n の時に上記不等式が成立すると仮定する。(これを「帰納法の仮定」という。)

f に対する条件から、$f(n+1) \leq 2f(n) + 2^{n+1} - 1$ である。ここで、帰納法の仮定は、$f(n) \leq n \cdot 2^n + 1$ であるので、これら 2 つを合わせて、$f(n+1) \leq 2(n \cdot 2^n + 1) + 2^{n+1} - 1$ となる。この式を整理すると、

$$f(n+1) \leq (n+1) \cdot 2^{n+1} + 1$$

が得られ、上記不等式が $n+1$ の時にも成立することが言えた。(step 終了)

以上より、n に関する帰納法により、上記不等式が任意の自然数 n に対して言えることが証明された。(証明終了)。

(解説) 上記の関数 f は、「長さ 2^n のリストを merge sort と呼ばれる方法で整列 (小さい順に並べかえ) したときの計算時間」を表す式である。上記のことから、merge sort が、長さ N のリストを $N \log N$ に比例する時間で (必ず) 整列するアルゴリズムであることが言える。(詳しくは、2 年生で学習する「データ構造とアルゴリズム」を参照のこと。)