関数プログラムの設計

胡 振江
東京大学 計数工学科
2006 年度

内容

- 関数プログラムの設計法
  - 解きたい問題を理解する.
  - 関数の型を決め，問題を解くための情報を集める.
  - 複雑問題を簡単な問題に分割して解く
- 再帰関数：Loop の表現
- 例題：平方根の計算

関数プログラムの設計

ステップ１： 解きたい問題を理解する.

例：三つの数字の中間値を求めよ.

- $2, 3, 4$ の中間値は 3 である.
- $2, 2, 4$ の中間値は 2？それともなし？

Thinking before doing!

ステップ２：関数の型を決め，問題を解くための情報を集める．

型はプログラムのインターフェース (入出力）を表す．

$$\mathsf{middleNumber} :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$$

有益な情報：

- 型上の基本関数：e.g., Int 上の比較演算など
- 既存の関数ライブラリ: e.g., 二つの値中から大きい方を返す関数 bigger

ステップ３：複雑問題を簡単な問題に分割して解く．

どのような関数があればこの問題を解けるのかを考えながら，問題を解く．

$$
\begin{aligned}
&\mathsf{middleNumber}\ x\ y\ z \\
&\quad \mid \mathsf{between}\ x\ y\ z\ =\ x \\
&\quad \mid \mathsf{between}\ y\ x\ z\ =\ y \\
&\quad \mid \mathrm{otherwise}\ =\ z
\end{aligned}
$$

$+$

m が n と p の中間値の時に True を返す関数
between m n p を設計する．

再帰定義

関数プログラミングでは，loop を再帰構造で表現する．

$$
\begin{array}{lll}
\mathsf{fac} :: & \mathsf{Int} \to \mathsf{Int} & \\
\mathsf{fac}\ n & \mid n == 0 & = 1 \\
& \mid n > 0 & = \mathsf{fac}\ (n-1) * n
\end{array}
$$

関数適用 $\mathsf{fac}\ 3$ の計算例：

$$
\begin{array}{rcl}
\mathsf{fac}\ 3 & \Rightarrow & (\mathsf{fac}\ 2) * 3 \\
& \Rightarrow & ((\mathsf{fac}\ 1) * 2) * 3 \\
& \Rightarrow & (((\mathsf{fac}\ 0) * 1) * 2) * 3 \\
& \Rightarrow & ((1 * 1) * 2) * 3 \\
& \Rightarrow & (1 * 2) * 3 \\
& \Rightarrow & 2 * 3 \\
& \Rightarrow & 6
\end{array}
$$

例：power2 $n = 2^n$ ：

$$
\begin{array}{ll}
\mathsf{power2} & :: \mathsf{Int} \to \mathsf{Int} \\
\mathsf{power2}\ n & \mid n == 0 \quad = 1 \\
& \mid n > 0 \quad = 2 * \mathsf{power2}\ (n-1)
\end{array}
$$

例：フィボナチー関数：

$$
\begin{array}{ll}
\mathsf{fib} & :: \mathsf{Int} \to \mathsf{Int} \\
\mathsf{fib}\ n & \mid n == 0 \quad = 0 \\
& \mid n == 1 \quad = 1 \\
& \mid n > 1 \quad = \mathsf{fib}\ (n-1) + \mathsf{fib}\ (n-2)
\end{array}
$$

例：最大公約数を求める関数 gcd:

$$
\begin{array}{lll}
\mathsf{gcd} & :: & \mathsf{int} \to \mathsf{Int} \to \mathsf{Int} \\
\mathsf{gcd}\ m\ n & | & n == 0 \qquad\quad = m \\
 & | & \mathsf{otherwise} \quad = \mathsf{gcd}\ n\ (\mathsf{mod}\ m\ n)
\end{array}
$$

例：最小公倍数を求める関数 lcm:

$$
\begin{array}{lll}
\mathsf{lcm} & :: & \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int} \\
\mathsf{lcm}\ m\ n & = & m * n\ /\ \mathsf{gcd}\ m\ n
\end{array}
$$

例：until 関数

$$\begin{aligned} \mathsf{until} &:: (a \to \mathsf{Bool}) \to (a \to a) \to a \to a \\ \mathsf{until}\ p\ f\ x &= \mathbf{if}\ p\ x\ \mathbf{then}\ x\ \mathbf{else}\ \mathsf{until}\ p\ f\ (f\ x) \end{aligned}$$

until の使用例：

$$\mathsf{until}\ (>100)\ square\ 2\ \Rightarrow\ 256$$

例題：平方根の計算

ステップ１： 解きたい問題を理解する.

問題

任意正整数 x と小さい数 ϵ に対して

$$\mathsf{sqrt}\ x \geq 0 \wedge |(\mathsf{sqrt}\ x)^2 - x)| \leq \epsilon.$$

を満たす sqrt 関数を作りたい.

ステップ２：関数の型を決め，問題を解くための情報を集める．

```
sqrt :: Float →  Float →      Float
        入力 x   小さい数 ε   結果
```

有益な情報：

- Float 上の関数：e.g., square, abs
- 既存の計算法：Newton 法

Newton 法

- 要求される精度に到達するまで繰り返して解の近似値を求める方法.
- y_n を x の平方根の近似値とすると

$$y_{n+1} = (y_n + x/y_n)/2$$

  による y_{n+1} は y_n よりもよい近似値である.
- 例：2 の平方根の計算：

$$\begin{array}{lcll} y_0 & = & 2 & \\ y_1 & = & (2 + 2/2)/2 & = 1.5 \\ y_2 & = & (1.5 + 2/1.5)/2 & = 1.4167 \\ y_3 & = & (1.4167 + 2/1.4167)/2 & = 1.4142157 \\ & \vdots & & \end{array}$$

ステップ３：複雑問題を簡単な問題に分割して解く.

$$\text{sqrt } x\ \epsilon \quad = \quad \text{until satisfy improve } x$$

ここで，小問題として satisfy と imrpove を次のように解くことができる.

$$\begin{aligned} \text{satisfy } y &= \text{abs}(\text{square } y - x) \leq \epsilon \\ \text{improve } y &= (y + x/y)/2 \end{aligned}$$

最終のプログラム

```
sqrt       ::  Float → Float → Float
sqrt x ε   =   let
                   satisfy y = abs(square y − x) ≤ ε
                   improve y = (y + x/y)/2
               in until satisfy improve x
```

注：シンタックスシュガー **where** を使って次のようにも定義できる.

```
sqrt       ::  Float → Float → Float
sqrt x ε   =   until satisfy improve x
               where
                   satisfy y = abs(square y − x) ≤ ε
                   improve y = (y + x/y)/2
```

Reference

- Simon Thompson, **Haskell: The Craft of Functional Programming** (Second Edition), Addison-Wesley, ISBN 0-201-34275-8, 1999.

  Chapter 4: Designing and Writing Programs.

- 教科書：2.1.5, 5.1