関数プログラミングの基本概念

胡 振江

東京大学 計数工学科

2008 年 10 月 6 日, 20 日

関数プログラミング

関数プログラミングは二つ部分からなる。

1. 関数を定義する。

$$square :: Integer \to Integer$$
$$square\ x = x * x$$

$$smaller :: (Integer, Integer) \to Integer$$
$$smaller\ (x, y) = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y$$

2. 計算機を用いて式を評価する。

```
?  square 3768
14197824
?  square (smaller (5,3+4))
25
?
```

関数プログラムの評価：式の簡約

関数プログラムの評価は，式を「最も単純な等価な形」に簡約し結果を表す過程である．

$$
\begin{array}{ll}
 & \mathit{square}\ (3+4) \\
\Rightarrow & \quad \{\ \text{def. of } +\ \} \\
 & \mathit{square}\ 7 \\
\Rightarrow & \quad \{\ \text{def. of square}\ \} \\
 & 7 * 7 \\
\Rightarrow & \quad \{\ \text{def. of *}\ \} \\
 & 49
\end{array}
$$

簡約方法は唯一ではない

式 $square\ (3+4)$ を簡単にする方法はこのほかにもある.

$$
\begin{array}{ll}
 & square\ (3+4) \\
\Rightarrow & \quad \{\ \text{def. of square}\ \} \\
 & (3+4)*(3+4) \\
\Rightarrow & \quad \{\ \text{def. of } +\ \} \\
 & 7*(3+4) \\
\Rightarrow & \quad \{\ \text{def. of } +\ \} \\
 & 7*7 \\
\Rightarrow & \quad \{\ \text{def. of *}\ \} \\
 & 49
\end{array}
$$

止まらない簡約もある

次のプログラムを考える。

$$three :: Integer \to Integer$$
$$three\ x = 3$$

$$infinity :: Integer$$
$$infinity = infinity + 1$$

式 *three infinity* の評価はどうなるか。

- infinity を優先的に簡約すると、無限ステップになる。
- three を優先的に簡約すると、結果がすぐに得られる。

関数プログラミング言語

関数プログラミング言語は，ラムダ計算 (lambda calculus) の概念をプログラミング言語として体現したもの.

代表的な関数プログラミング言語

- Lisp 1958- (Scheme 1970 年代後半-)
- ISWIM 1966-
- ML 1970 年代後半-
- Miranda 1985-
- **Haskell** 1987- (純粋関数プログラミング言語，Haskell の名前は論理学者 Haskell B. Curry に由来, 本講義で使う言語)

関数（数学）

入力 x に対して，出力 y のただ一つの値を決定する規則が与えられているとき、 y を x の関数という．対応規則を明示するときは，

$$y = f(x)$$

のように対応規則に名前を付与する．

C の関数 $\neq$ 関数 (数学)

例：プログラミング言語 C の関数定義

```
1: int y = 10;
2: int f (x: int) {
3:     printf ("Hello!");
4:     return y * x;
5: }
```

次の点で数学の関数とは異なる.

- 副作用を持つ：関数の処理の実行によってシステムに変化が発生する
- 状態を持つ：引数が同じでも状況に応じて戻り値が異なる

本講義中の関数 = 関数 (数学)

関数

$$f :: A \to B$$

は型 A の引数 (argument) をとり，型 B の結果 (result) を返すという．また，x が A の元を表しているとき，x に関数 f を適用した結果を $f(x)$ または $f\ x$ で表す．

$$\begin{aligned} &\mathit{square} :: \mathit{Integer} \to \mathit{Integer} \\ &\mathit{smaller} :: (\mathit{Integer}, \mathit{Integer}) \to \mathit{Integer} \\ &\mathit{three} :: \mathit{Integer} \to \mathit{Integer} \end{aligned}$$

関数の外延性

関数の相等

$f = g$ iff 任意の x に対して、$f\ x = g\ x$ が成り立つ。

次の二つの関数が等しい。

$$double, double' :: Integer \to Integer$$
$$double\ x = x + x$$
$$double'\ x = 2 * x$$

カリー化

カリー化

構造をもつ値の引数を単純な引数の列に置き換える方法。

- 引数が構造を持つ

$$\begin{aligned} &\mathit{smaller} :: (\mathit{Integer}, \mathit{Integer}) \to \mathit{Integer} \\ &\mathit{smaller}\ (x, y) = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y \end{aligned}$$

- 引数が構造を持たない

$$\begin{aligned} &\mathit{smallerc} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer} \\ &\mathit{smallerc}\ x\ y = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y \end{aligned}$$

カリー化の利点：部分関数

関数定義

$$\begin{aligned} &\mathit{smallerc} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer} \\ &\mathit{smallerc}\ x\ y = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y \end{aligned}$$

は

$$\begin{aligned} &\mathit{smallerc} :: \mathit{Integer} \to (\mathit{Integer} \to \mathit{Integer}) \\ &(\mathit{smallerc}\ x)\ y = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y \end{aligned}$$

と同じである。従って、

$$\mathit{smallerc}\ 5$$

は関数（どうな関数？）を表している。

二項演算子のセクション表現

セクション：括弧でくくられた演算子

$$\begin{array}{lcl} (+) & :: & \mathsf{Int} \to \mathsf{Int} \to \mathsf{int} \\ (+)\ x\ y & = & x + y \end{array}$$

括弧でくくられた infix 演算子が普通の prefix 関数のように引数を適用することができる．また，引数として関数に渡したりすることができる．

$$\mathsf{both}\ f\ x = f\ x\ x$$

と定義すると，

$$\mathsf{both}\ (+)\ 3 \ \Rightarrow\ (+)\ 3\ 3 \ \Rightarrow\ 3+3 \ \Rightarrow\ 6$$

二項演算子のセクション表現

更に拡張： 引数を演算子とともに括弧でくくる.

$$
\begin{aligned}
(x\oplus)\ y &= x \oplus y \\
(\oplus y)\ x &= x \oplus y
\end{aligned}
$$

例：

$(*2)$: ２倍する関数
$(1/)$: 逆数を求める関数
$(/2)$: ２分する関数
$(+1)$: つぎの値を得る関数

関数の合成

関数合成

$$(.) :: (\beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$$
$$(f\ .\ g)\ x = f\ (g\ x)$$

例：

$$quad :: Integer \to Integer$$
$$quad = square\ .\ square$$

性質

$$id\ .\ f = f\ .\ id = f$$
$$(f\ .\ g)\ .\ h = f\ .\ (g\ .\ h)$$

型

型 (type) はある種類の値の集まりである.

- 基本型
  - 整数型 Integer/Int: $\{\ldots, -3, 2, 0, 10, \cdots\}$
  - 浮動小数点数型 Double/Float: $\{\ldots, -3.14, 3, 0, 123.45, \ldots\}$
  - 論理型 Bool: $\{True, False\}$
  - 文字型 Char: $\{'a', 'b', \ldots, 'A', 'B', \ldots\}$
  - 文字列型 String: $\{\text{“}HelloWorld\text{”}, \ldots\}$
- 合成型 (派生型)
  - リスト型 $[t]$: $\{[], [2, 5, 4], \ldots\}$
  - 組型 (t_1, t_2): $\{(1, 2), ('a', 3), \ldots\}$
  - 関数型 $t_1 \rightarrow t_2$: $\{square, smallerc\ 5, \ldots\}$

関数の型・多様型

関数プログラミングにおいて，関数は値であり，他の値と対等のものである．関数を引数として関数に渡したり，結果として返したりする．

$$apply :: (\alpha \to \beta) \to \alpha \to \beta$$
$$apply\ f\ x = f\ x$$

$$curry :: ((\alpha, \beta) \to \gamma) \to \alpha \to \beta \to \gamma$$
$$curry\ f\ x\ y = f\ (x, y)$$

$$uncurry :: (\alpha \to \beta \to \gamma) \to (\alpha, \beta) \to \gamma$$
$$uncurry\ f\ (x, y) = f\ x\ y$$

関数の定義 (1/3)

条件の表現：

$$
\begin{aligned}
&\mathit{smaller} :: (\mathit{Integer}, \mathit{Integer}) \to \mathit{Integer} \\
&\mathit{smaller}\ (x, y) = \textbf{if}\ x \leq y\ \textbf{then}\ x\ \textbf{else}\ y
\end{aligned}
$$

または

$$
\begin{aligned}
&\mathit{smaller} :: (\mathit{Integer}, \mathit{Integer}) \to \mathit{Integer} \\
&\mathit{smaller}\ (x, y) \\
&\quad \mid x \leq y \ = x \\
&\quad \mid x > y \ = y
\end{aligned}
$$

関数の定義 (2/3)

再帰の表現：

$$fact :: Integer \to Integer$$
$$fact\ n = \textbf{if}\ n == 0\ \textbf{then}\ 1\ \textbf{else}\ n * fact\ (n-1)$$

評価例：

$$\begin{aligned}
& fact\ 1 \\
= & \quad \{\ \text{def. of fact}\ \} \\
& \textbf{if}\ 1 == 0\ \textbf{then}\ 1\ \textbf{else}\ 1 * fact\ (1-1) \\
= & \quad \{\ \text{since } 1 == 0 \text{ evaluates to False}\ \} \\
& 1 * fact\ (1-1) \\
= & \quad \{\ 1-1=0\ \} \\
& 1 * fact\ 0 \\
= & \quad \{\ \text{many steps ...}\ \} \\
& 1
\end{aligned}$$

関数の定義 (3/3)

局所定義：

$$
\begin{aligned}
&f :: (Float, Float) \rightarrow Float \\
&f\ (x, y) = (a + 1) * (b + 2) \\
&\quad \textbf{where} \\
&\qquad a = (x + y)/2 \\
&\qquad b = (x + y)/3
\end{aligned}
$$

または、

$$
\begin{aligned}
&f :: (Float, Float) \rightarrow Float \\
&f\ (x, y) = (a + 1) * (b + 2) \\
&\quad \textbf{where}\ a = (x + y)/2;\ \ b = (x + y)/3
\end{aligned}
$$

関数プログラムの処理系

本講義で関数プログラミング言語 Haskell (コア部分)
http://www.haskell.org/haskellwiki/Haskell
を使う.

- Haskell のコンパイラ: ghc
  http://www.haskell.org/ghc/
- Haskell のインタプリタ: hugs
  http://haskell.org/hugs/

Haskell のインタプリタ: Hugs

- Mark P. Jones らによって開発された Haskell のインタプリタで，現在の最新版は Hugs 98 である.
- Hugs は，多くの Unix や Windows 上で動くことが確認されており，コンパイル済のバイナリコードが Win32, Linux, Machintosh の各プラットフォームに用意されている.
- Windows 用の Hugs (winhugs) は GUI を備えており，初心者でも扱いやすくなっている.
- Hugs のインストルは簡単である.

宿題

- 教科書第一章を復習し、演習問題をやる。
- Hugs または ghc システムをインストルする。
- 「Learning Haskell in 10 Minutes」を自習する。

  http://www.haskell.org/haskellwiki/Learn_Haskell_in_10_minutes