プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 2a

目次

プログラム言語の種類

計算モデルに関する分類

- 手続き型言語 (Procedural PL): C 言語など非常に多数
- 宣言型言語 (Declarative PL)
  - 関数型言語: Lisp, Scheme, OCaml, Haskell など
  - 論理型言語: Prolog など

そのほかの分類としては、たとえば、関数型、論理型、オブジェクト指向型などの「パラダイム」に関する分類など。

目次

関数型プログラミング言語

関数 (正確には、部分関数) の概念を中心に置いたプログラミングを行うための言語である。

- 手続き型言語における計算==「状態を更新して、次の状態にする手続き」
- 関数型言語における計算==「引数から返り値を計算する手続き」

C 言語の文 (statement): `x = x+3;` という文は、(x=n) という状態を (x=n+3) という状態に更新する手続きをあらわす。

ML 言語の式 (expression): `fun x -> x*3` という式は、整数 n をもらうと $n+3$ を返す関数をあらわす。

代表的な関数型プログラム言語

- Lisp (Common Lisp など多数の種類あり)
- Scheme: Lisp の一種と見なすことが多い
- ML 系言語 (SML, OCaml, F# など)
  - SML... 正式には Standard ML (最初の ML 言語)
  - OCaml ... もともと Objective Caml だった。(Caml 言語にオブジェクト指向機能を追加した。)
- Haskell: 「純粋」な関数型言語

なぜ、関数型プログラム言語を学習するのか

仮定: この授業を受けている人は、「最初のプログラミング言語」として、Python,C,Java 等の言語を習っているはず。→手続き型言語の基礎はわかっているだろう。
なぜ関数型言語に基づいてプログラミング言語を考えるのか？

- 手続き型言語やオブジェクト指向言語より、関数型言語の方が簡単だから。
  (関数型言語の型システムを理解したあと、オブジェクト指向言語などの、より複雑な型システムを理解することはできるが、逆はやりにくい。)
- 型理論の発展の歴史は、関数型言語の発展の歴史でもあるから。

参考. この授業では、最後の方で、Java (オブジェクト指向言語) の型システムについて学習する。

目次

OCaml プログラミング (1)

いろいろな関数の定義 (キーワード let で始める)

```
let f1 x = x*x
let f2 x = (f1 (x+1) + f1 (x-1)) / 2
let f3 x y = f1 (x + (f2 y))
let f4 x = if x>10 then 10 else x
```

関数の利用

```
let ex1 = f3 5 10
(*  ==> 11236 *)
let ex2 = (f4 5) + (f4 15)
(*  ==> 15 *)
```

OCaml プログラミング (2)

再帰関数の定義 (キーワード let rec で始める)

```
let f5 x = if x=0 then 1 else f5(x-1)+2
(*  ==> エラー  *)
let rec f5 x = if x=0 then 1 else f5(x-1)+2
let ex3 = f5 10
(*  ==> 21  *)
```

OCaml プログラミング (3)

最大公約数を計算する関数

```
let rec gcd x y =
    if x=0 then y
    else if y=0 then x
    else if x>y then gcd (x-y) y
    else gcd (y-x) x

let ex4 = gcd 20 3
(*  ==> 1  *)
let ex5 = gcd 20 4
(*  ==> 4  *)
```

OCaml プログラミング (4)

Fibonacci 数を計算する関数

```
let rec fib n =
    if n<=1 then 1
    else (fib(n-2)) + (fib(n-1))
let ex6 = fib 10
(*  ==> 89  *)
let ex7 = fib 40
(*  とても時間がかかる  *)
```

「組 (tuple)」のデータ構造を利用

```
let rec fib2 n =
    if n=0 then (1,1)
    else let (a,b) = fib2(n-1) in
           (b, a+b)
let ex8 = fib2 40
(*  ==> (165580141, 267914296)  *)
```

OCaml プログラミング (5)

再帰の利用

```
(* 階乗の計算 *)
let rec factorial n =
    if n=0 then 1
    else n * (factorial(n-1))
let ex = factorial 10
(*  ==> 3628800 *)
(* べき乗の計算 *)
let rec power x n =
    if n=0 then 1
    else x * (power x (n-1))
let ex = power 2 10
(*  ==> 1024  *)
```

OCaml プログラミング (番外編 1)

再帰でなく繰返しを使う (この授業では推奨しない)

```
(* べき乗の計算 *)
let power2 x n =
  let k   = ref n in
  let res = ref 1 in
  while (!k > 0) do
    res := !res * x;
    k   := !k - 1
  done;
  !res    (* この値を返す *)
let ex = power2 2 10
(*  ==> 1024  *)
```

OCaml プログラミング (番外編 2)

再帰ではあるが繰返しのように関数を呼び出す

```
(* べき乗の計算 *)
let rec power3 x (n, res) =
  if n=0 then res
  else power3 x (n-1, res*x)
let ex = power3 2 (10,1)
(*  ==> 1024  *)
```

power3 の第 2 引数が、
(10,1)=>(9,2)=>(8,4)=>(7,8)=>...=>(0,1024) と変化し、繰返し処理と実質的に同じ。

OCaml プログラミング (6)

Collatz 予想 (未解決問題):

```
let rec collatz n =
    if n=1 then 1
    else if n mod 2 = 0 then collatz(n/2)
    else collatz(n*3+1)
let ex = collatz 100
(*  ==> 1  *)
```

OCaml プログラミング (7)

関数もデータとなる (高階関数)

```
let compose f g x = f (g x)
let add1 x = x+1
let multiply3 x = x*3
let ex1 = compose add1 add1 10
(*  ==> 12 *)
let ex2 = compose add1 multiply3 10
(*  ==> 31 *)
```

関数を n 回呼び出す (高階関数)

```
let rec iterate f x n =
  if n=0 then x
  else iterate f (f x) (n-1)
let ex3 = iterate add1 10 5
(*  ==> 15 *)
let ex3 = iterate multiply3 10 5
(*  ==> 2430 *)
```

型付けについて少しだけ

OCaml は「静的」に「型付け」する言語である。

- 静的 (static): 実行時 (動的) より前、通常は「コンパイル時」のこと。
- 型付け (typing): 式やプログラムに、「型」を割り当てること。

これまでの関数の型:

```
f1,f2,f3,f4,f5    : int → int
fib,fib2,collatz  : int → int
factorial         : int → int
gcd, power        :  int → int → int
```

高階関数 (`compose` と `iterate`) の型は難しいので後述。

まとめ

- 関数型プログラミング言語は、「引数から返り値への (部分) 関数」を組み合わせる形でプログラムを作成する。
- 関数の中には、「他の関数をデータとして受けとる」関数もあり、高階関数 (higher-order function) と呼ばれる。
- OCaml 言語は関数型プログラミング言語の 1 つであり、静的に型付けされる。