プログラム言語論

亀山幸義

筑波大学 情報科学類

関数型言語の補足

複数の引数をもつ関数

In C:

```
void foo (int x, int y) {
  return x + y;
}
```

In OCaml: (2 つの書き方がある。)

```
let foo (x,y) = x + y in
  foo (10,5)
   ==> 15

let foo x y = x + y in
  foo 10 5
   ==> 15
```


複数の引数をもつ関数

1. 組 (タプル) を使って、複数の値を 1 つにまとめる

```
let foo (x,y) = x + y ;;
```

`(10,5)` は 1 つの値。

2. 高階関数を使う。

```
let foo x y = x + y ;;
```

`foo 10` は 1 つの値。
`(foo 10) 5` も、また、1 つの値。
これは、`foo 10 5` と書いても同じ。(関数適用において、括弧を省略すると、左から 結合する。)

カリー化された関数

第二の形式 (高階関数の利用) を「カリー化された関数」と呼ぶ。

カリー化 (currying): 第一の形式を第二の形式にすること。

非カリー化 (uncurrying): 第二の形式を第一の形式にすること。

```
let foo x y = x + y;;
   ==> val foo : int -> int -> int = <fun>
foo 10;;
   ==> - : int -> int = <fun>
(foo 10) 5;;
   ==> - : int = 15
```

関数 `foo` の型は、`(int * int) -> int` ではなくて、
`int -> (int -> int)` になる。

型 `int -> (int -> int)` は、型 `int -> int -> int` と同じである。(型の `->`に関しては、括弧を省略した場合、右から結合する。)

カリー化された関数の例

利用する側では、引数を自然に並べればよい。

```
let foo x y z = x + y + z in
  (foo 1 2 3) * (foo 5 6 7)
```

あるいは、ラムダ式を使って書いてもよい (同じ)。

```
let foo' =
  fun x -> fun y -> fun z -> x + y + z in
  (foo 1 2 3) * (foo 5 6 7)
```

ただし、引数の個数が間違っていると、変なことになる。

```
let foo x y z = x + y + z;;
(foo 1 2) ;;     (* エラーなし; 部分適用 *)
(foo 1 2) * (foo 5 6 7)  ;; (* 型エラー *)
```


カリー化のまとめ

高階関数を使うと、組を使わずに複数の引数をもつ関数が表現できる。

- 組を使う必要がない。プログラムが見た目にきれいになる。
- 部分適用が可能となる。

```
let add x y = x + y in
let f = add 10 in
  (f 5) * (f 20) * (f 6)
```

- 関数クロージャを生成してしまう処理だと、実行性能が悪い。(problem)

problem に対する対処 (多くの関数型言語の処理系):

- 関数適用の処理を頑張っていて、(部分適用でなければ) 関数クロージャを生成しないようにしている。(e.g. OCaml 処理系の基礎の 1 つである Zinc Abstract Machine、主専攻実験 S8 のテキストを参照)