関数型言語（OCaml 演習）(14)

値の書き換えと参照透過性

浅井 健一

お茶の水女子大学

参照透過性

これまでのプログラム：

- データは不変でいつも同じ値を持つ。
- 関数は、同じ引数なら、必ず同じ結果を返す。

これまでのデータ、関数は、参照透過性を持つ。

データ、関数が参照透過性を持つと

- 「現在の値」を意識する必要がない。
- 関数の動きは、入力のみを考えれば理解できる。

特段の理由がないなら、参照透過性は保つべき。

呼び出し回数のカウント

次の関数 `fib` を `fib 8` のように呼び出すと（再帰呼び出しも含めて）全体で `fib` は何回、呼び出されるか。

```
(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =

  if n < 2 then n
  else fib (n - 1) + fib (n - 2)
```

フィボナッチ数列：0 1 1 2 3 5 8 13 21 34 55 89 144 ...

呼び出し回数のカウント

次の関数 fib を fib 8 のように呼び出すと（再帰呼び出しも含めて）全体で fib は何回、呼び出されるか。

カウンタを 0 にセットする

```
(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =
 (カウンタを 1 増やす;
   if n < 2 then n
   else fib (n - 1) + fib (n - 2))
```

フィボナッチ数列：0 1 1 2 3 5 8 13 21 34 55 89 144 ...

書き換え可能なセル

ref 初期値 （型は 'a ref）

書き換え可能なセル（箱）を作り、初期値を設定する。

```
# ref 3 ;;
- : int ref = {contents = 3}
# let counter = ref 0 ;;
val counter : int ref = {contents = 0}
```

! セル

セルの中身を返す。

```
# !counter ;;
- : int = 0
```

セルの書き換え

セル := 値

セルの中身を値で書き換え、() を返す。

```
# !counter ;;
- : int = 0
# counter := 1 ;;
- : unit = ()
# !counter ;;
- : int = 1
```

!counter の結果は「現在のセルの値」に依存する。

呼び出し回数のカウント

カウンタを 0 にセットする

```
(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =
 (カウンタを 1 増やす;
  if n < 2 then n
  else fib (n - 1) + fib (n - 2))
```

呼び出し回数のカウント

```
let counter = ref 0

(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =
 (counter := !counter + 1;
  if n < 2 then n
  else fib (n - 1) + fib (n - 2))
```

呼び出し回数のカウント

```
let counter = ref 0

(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =
 (counter := !counter + 1;
  if n < 2 then n
  else fib (n - 1) + fib (n - 2))

# fib 8 ;;
- : int = 21
# !counter ;;
- : int = 67
```

呼び出し回数のカウント

```
let counter = ref 0

(* 目的：フィボナッチ数を求める *)
(* fib : int -> int *)
let rec fib n =
 (counter := !counter + 1;
  if n < 2 then n
  else fib (n - 1) + fib (n - 2))
```

```
# fib 8 ;;
- : int = 21
# !counter ;;
- : int = 67
```

```
# fib 8 ;;
- : int = 21
# !counter ;;
- : int = 134
```

副作用命令の影響範囲

副作用を使わない関数

- 動きは入力のみに依存。
- 入力が同じなら、出力も同じ。

セルを使う関数

- 動きは入力と現在のセルの値に依存。
- 入力が同じでも、セルの値によって結果が変わる。

セルを導入すると常にセルの値を意識する必要がある

OCaml では、値の変更を `'a ref` 型のデータや配列など一部の型に限定することで、関数型の良さを失うことなく、副作用を導入している。

配列

書き換え可能なデータが並んだもの。先頭を 0 として順に番号がふられており、その番号を使って任意のデータに定数時間でアクセスできる。

[| 式$_0$; 式$_1$; ...; 式$_{n-1}$ |]

長さが n の配列を作る。

```
# let a = [| 3; 5; 2; 4 |] ;;
val a : int array = [|3; 5; 2; 4|]
```

配列の中のデータは、みな同じ型でなくてはならない。

配列へのアクセス

Array.get 配列 添字
配列.(添字)

配列の添字番目をとってくる。

```
# let a = [| 3; 5; 2; 4 |] ;;
val a : int array = [|3; 5; 2; 4|]
# Array.get a 3 ;;
- : int = 4
# a.(1) ;;
- : int = 5
# a.(4) ;;
Exception: Invalid_argument "index out of
bounds".
```

配列の書き換え

Array.set 配列 添字 値
配列.(添字) <- 値

配列の添字番目を値に変更し、() を返す。

```
# let a = [| 3; 5; 2; 4 |] ;;
val a : int array = [|3; 5; 2; 4|]
# Array.set a 1 7 ;;
- : unit = ()
# a ;;
- : int array = [|3; 7; 2; 4|]
# a.(2) <- 6 ;;
- : unit = ()
# a ;;
- : int array = [|3; 7; 6; 4|]
```

まとめ

- セル（'a ref 型）。

  ref 初期値　　! セル　　セル := 値

- 配列（'a array 型）。

  [|...|]

  Array.get a 3　　a.(3)

  Array.set a 2 6　　a.(2) <- 6

参照透過性

データは不変でいつ見ても同じ値を持っていること。

参照透過性を持たないデータを扱うには常に「現在の値」を意識してプログラムを作る必要がある。