プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 12

この資料の位置付け

この資料は、学生から質問された項目や、授業での自分の説明がわかりにくかった点などを補足するものです。

ML の関数の引数についての補足 (0)

OCaml の関数の型についての質問があった。

```
let f x = x + 1;;
  val f : int -> int = <fun>

let g x y = x + y;;
  val g : int -> int -> int = <fun>
```

関数 f の型が `int -> int` であることは納得できるが、なぜ、関数 g の型は `int -> int -> int` なのか、わかりにくい、と言われた。

ML の関数の引数についての補足 (1)

```
let g x y = x + y;;
  val g : int -> int -> int = <fun>
```

実は‥、

```
let g x = fun y -> x + y;;
let g = fun x -> fun y -> x + y;;
let g = fun x -> (fun y -> x + y);;
```

- 関数 g は、int 型の x を受け取り、`fun y -> x + y` という関数を返す。(高階関数)
- ところで、関数 `fun y -> x + y` は、int 型の y を受け取り、`x + y` という int 型の値を返しているので、`int->int` という型を持つ。
- よって、関数 g は、`int -> (int->int)` という型を持つ。
- 型 `int -> int -> int` は、型 `int -> (int -> int)` の事である。(型の世界では、右にあるものが優先的に括弧付けられる。右結合的)

ML の関数の引数についての補足 (2)

カリー化 (Currying) ·· 2 引数の関数のかわりに、1 引数の高階関数を使うこと。(あるいは、2 引数の関数を、1 引数の高階関数に変換すること。)

```
let g_uncurry (x,y) = x + y;;
  val g_uncurry : int * int -> int = <fun>

let g_curry x y = x + y;;
  val g_curry : int -> int -> int = <fun>
```

g_uncurry (カリー化されていない関数、普通のプログラム言語でよく見る関数) は、int * int 型の値 (整数値 2 つがペアになっている値) を 1 つもらって、int 型の値を 1 つ返す関数。
g_curry (カリー化された関数、関数型言語でよく見る関数) は、int 型の値を 1 つもらって、「int 型の値を 1 つもらって、int 型の値を 1 つ返す関数」を返す高階関数。

ML の関数の引数についての補足 (3)

g_curry は、2 引数関数ではなく、カリー化された高階関数なので、以下のようなこともできる。引数を 1 つだけ与えることもできる。

```
let g x y = x + y;;
let h = g 10 in
  (h 20) + (h (h 30));;
==> 80
```

g 10 は、fun y -> 10 + y という関数になる。

Polymorphism についての補足 (0)

この授業では、Polymorphism という言葉が何度か、異なる意味で出てきており、授業中では「それが違う」としか言わなかったので、ここで 3 つのケースに分類して整理する。
まず、Polymorphism の基本的な意味は、「1 つの関数 (あるいはメソッド、式など) が、様々な型をもつ引数に適用できる」という意味であり、このことは 3 つのケースで共通である。

Polymorphism についての補足 (1)

(1) 関数型プログラム言語等で、「関数を表す式が Polymorphic Type を持つ」というときの Polymorphism (Parametric Polymorphism)

- この場合 Polymorphism を「多相性」と訳す。
- 1 つの関数が、「あるパターンにあてはまる任意の型」をもつ引数に適用できる、という意味。
- 例: fun x -> x という関数は int -> int 型も、bool -> bool 型も持つ。一般に、'a -> 'a 型を持つ。
- ここで 'a は型変数 (型パラメータ) であり、任意の型を表す。
- 別の例: fun f -> fun x -> f (f x) という式は ('a -> 'a) -> ('a -> 'a) 型をもつ。

Polymorphism についての補足 (2)

(2) オブジェクト指向言語で、「型 A の引数を持つメソッドは、型 A の任意のサブタイプ B の要素に適用できる」というときの Polymorphism (Subtype Polymorphism)

- オブジェクト指向言語では、Polymorphism を「多態性」と訳すことが多い。
- 1 つのメソッド (あるいは関数、式など) が型 A のオブジェクトに適用できるとき、「その型 A の任意のサブタイプ」をもつ引数にも適用できる、という意味。
- 例: この授業の Java 演習で、Point 型のオブジェクトを書いてよい場所には、ColoredPoint 型のオブジェクトを書いてもよい。
- 別の例: Java の演習で用いた `toString` メソッドは、Point 型の要素にも ColoredPoint 型の要素にも適用可能である。(override により、これらの 2 つの型において `toString` メソッドの本体は異なる.)

Polymorphism についての補足 (3)

(3) オブジェクト指向言語や一部の関数型言語にある overload の機構も一種の Polymorphism である (Ad Hoc Polymorphism)

- 1 つのメソッド (あるいは関数、式など) を、異なる型を持つ引数 (あるいは、引数の個数自身が違う場合など) に適用できる、という意味。
- 例: Java の overloading. Java 演習で用いた、foo メソッドは、引数が 1 個か 2 個か、また、それらの型が何かによって、異なる定義となっていた。
- 別の例: 多くのプログラム言語で、+ という記号は、「整数同士の加算」と「実数同士の加算」の両方の意味を持つ。Java では「数の足し算」以外に「文字列の連接」操作もあらわす。