



プログラミング言語論 #04

Product of Types

2018-05-07

Haskell 言語を用いて型宣言を簡単に記述し，関数の型について
考察するための道具を得た. これを基に型の観点から個々のプロ
グラミング言語の持つ限界について考察する. 今日は返値に関する
制約について考えることで新たな型を導出する.

講義で使用する言語

プログラミング言語論では型の面からプログラミング言語の進化、特徴を比較する。対象として以下の言語を主として用いる。

解答にこれくらいは列挙されていることを期待しているということ。

C 古典代表 [▶ repl.it](#)

C++ 産業界で重要 [▶ repl.it](#)

Haskell 研究で人気 [▶ repl.it](#)

Java 産業界で重要 [▶ repl.it](#)

JavaScript 毛色の違う例 [▶ repl.it](#)

Python 様々な分野で人気 [▶ repl.it](#)

これらについては「XXX 言語」ではなく「XXX」と記述する。

以下の言語について興味（知識）がある人は自分で調べた上で例示に挙げても全く構わない。

C# Go, Lisp 族, Objective-C, Ocaml, PHP, Ruby, Scala, Smalltalk 族, Swift

限界の例

- 返値の個数 – なぜ 1 つしか返せないのか
- 仮引数の個数の上下限 – 1 個の引数はあり得るか
- 実引数と仮引数の個数の一致 – 必ず一致しなければならないのか
- 「A または B」型はあるのか – 集合論的な進化
- 「A 以外」型はあるのか – 集合論的な進化
- 2 重定義とは何か
- 型が再帰できるか

考えてもしようがないものも含む

返値の数について考えよ

1 f :: T1 -> T2 -> -> Tn -> Tr

- ① この中で返値はどれか
- ② C で Tn と Tr (または Tr と Ts) を返せるようにせよ
- ③ Haskell で Tn と Tr (または Tr と Ts) を返せるようにせよ

成績計算関数

```
1 // 氏名nameの順位 :: int と平均点 :: double を返す関数
2 int double getSeiseki (char *name) {
3     junniとheikintenの計算には非常に時間が掛かる
4     return junni;
5     return heikinten;
6 }
```

- 構文的な理由でできないから構文を拡張する
- 型的な理由でできないから型を拡張する
- どうやってもできない

複数の型から一つの型を構成する

- 複素数 : $3 + 2i, -1 - 5i$ – 実部と虚部から 1 つの型 Z を構成
- 3 次元座標 : $p = (x, y, z)$ – 3 つの座標値から 1 つの値 p を構成

極めて一般的な要求なので言語の基本機能として提供すべき

Haskell の場合

```
1  a ::  
2  a = (1, 2)  
3  
4  b ::  
5  b = (1 + 3, mod 10 3, 0, -8, -1)  
6  
7  c ::  
8  c = ("AA", 10, False)  
9  
10 d ::  
11 d = ((0, "zero"), (1, "one"))
```

複数のものをカンマで区切り括弧でまとめるという記法は C の関数呼出しの引数部分と同一。 Haskell ではこの機能を導入したかったので関数呼出しに括弧もカンマも使わないことにした。

Haskell での実例

定義に合わせ宣言を埋めよ

```
1 f1 ::  
2 f1 x = (x, x+1, x+2)  
3  
4 f2 ::  
5 f2 x = (x, x == "0")  
6  
7 f3 ::  
8 f3 x = (60 <= x, 80 <= x)
```

ライブラリ関数で既に利用されている

```
1 div      :: Int -> Int ->  
2 mod      :: Int -> Int ->  
3  
4 divMod  :: Int -> Int ->           -- 別々に計算するよりお得
```

構文を拡張することなく複数の値を返すという目的が達成できた

別の形で利用してみる

この機能は新しい構文ではなく型として定義された。従って他の型と同様の使いができるはずである。

返値ではなく引数として利用

```
1 f4 ::  
2 f4 z = z == (1, False)  
3  
4 f4' ::      ->      ->  
5 f4' x y = (x == 1) && (y == False)
```

役に立つか：（構文を拡張することなく）引数の個数を減らせる

- パラメータが多い関数の簡単化（線を画面に書く関数）
- 3次元座標の計算が $x + y$ と書ける（かも）
- 手紙が 2通届くのと手紙が 2通入った小包を 1回受け取るのの違い

The objectives
oo

Tuple as a product of objects
oooo●oo

A new era
o



言語の違い

C++, Haskell, Python

言語仕様に組み込まれている：組またはタプルと呼ぶ。

C

構造体を使えばよい。ただしわざわざ構造体を定義する必要がある。または `scanf` のようなことをする（詳細省略）。

JavaScript

ない。しかし困らない：後述

Java

ない。裏技はある：後述

蛇足：これでもまだ対称性がないという人へ

返値だけまとめるのはカッコ悪い

関数のあるべき姿（案）

1 $f :: (a_1, a_2, a_3, \dots) \rightarrow (r_1, r_2, r_3, \dots)$

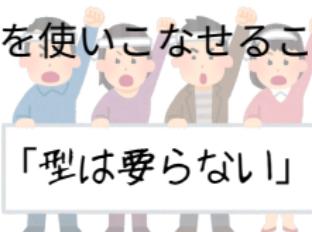
これこそが対称. 全ては 1 引数 1 返値関数に統一された.

しかし便利かどうかは疑問が残る.

そのうち分かるが, こうかいするかもしれないかも.

言語における型システムの意義

- このように型に関する設計・仕様を柔軟なものにすることで、言語に対する要求を新言語を作ることなく達成することができる
- 新しい言語ほど型に関する研究成果が取り入れられている
- プログラマは型を使いこなせることが必要
- その中でむしろ「型は要らない」という一派も出てきた
 - ▶ JavaScript
 - ▶ Python



これは意訳しすぎ。正確な主張は以降の資料にて（明示はしていないかも）