プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 6: 型システムその 1 (基本)

型 (Type) とは?

「型」は「データの集合」の一種ではあるが、データの集合がすべて型になるとは限らない。

- コンピュータ (ハードウェア) で扱うことのできるデータの種類 のこと。
- 同じ演算が適用できるデータの集まり。

型システム： どのようなプログラムにどのような型がつくか、定めるための体系。プログラム言語の設計における最重要要素の 1 つ。
静的な型システム: 型がつくかどうかを、静的に (プログラム実行前に) チェックして、型がつくプログラムのみをコンパイル・実行する。
動的な型システム: 型がつくかどうかを、動的に (プログラム実行時に) チェックして、型がつかないことがわかったら実行時エラーとする。

microML と OCaml の違い

例 (OCaml の構文で記述する)：

```
1 + true
10 (fun x → x + 1)
fun x → x x
```

これらは、microML では書けるが、1-2 個目は実行時エラー、3 個目は後で使おうとするとエラーになる。

OCaml ではそもそも書けない (コンパイルエラー)。

→ OCaml の方がうれしい。(cf. ソフトウェア工学の大原則; 不具合は、なるべく上流で発見したい)

Type System

B. Pierce, "Types and Programming Languages", MIT Press, 2002. (型理論の著名な教科書)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

- プログラムの (良くない) 振舞いが「ない」ことを証明する。
- 扱い可能な (現実的な) 構文的な方法
- (プログラムの) フレーズを分類する
- (フレーズたちが、それぞれ) 計算する値の種類

具体的な型

基本型 (atomic type)

- 例: `int`, `bool`, `string`, ...

複合的な型: 既にある型と型構成子 (type constructor) を使って構成。

- 例
  - C 言語: 構造体 (strucut)、共用 (union)、ポインタ、関数など。
  - Java 言語: クラス (オブジェクトの型) など。
  - ML 言語: 直積、レコード (record)、バリアント (variant), 参照、関数、リスト、再帰的な型など。
  - Lisp/Scheme 言語: S 式 (S expression)

C 言語の型の例

```
void foo (int *p, int *q) {
   while (*p) {
     *(q++) = *(p++);
   }
}
```

C 言語は静的型付け: コンパイル時に型検査を行なう。

- 上記関数の 3 行目を `q++ = *p++;` にするとコンパイル・エラー
- cf. Scheme などの動的型付け言語では、型エラーを含む関数定義があっても、コンパイル時にはエラーにならない。(それが実際に実行されるときにエラーになる。)

OCaml の型 (1)

現代のプログラミング言語は、豊富なデータ型を用意するものが多い。

1. 基本型

```
10    : int
true  : bool
"abc" : string
let x = 10 in x * 2 + 3 : int
(fun x → x * 3 = 10) (5 + 1) : bool
```

2. 直積型 (組 tuple が持つ型)$A * B$、リスト型 A *list*

```
(10, 20, "abc") : int * int * string
[10; 20; 30] : int list
[[10; 20]; [30; 40; 50]] : (int list) list
```

他の言語ではリスト型を $List(A)$ といった表記のものもある。

OCaml の型 (2)

3. 関数型 ($A \to B$)

```
fun x → x + 1 : int → int
fun x → (x+1, x−1) : int → (int * int)
fun x → [1; 2; 3] @ x @ [4; 5] : (int list) → (int list
    )
```

高階関数

```
fun f → (f 10) * 5  : (int → int) → int
fun x → (fun y → x + y) : int → (int → int)
fun f → (fun x → (f(x+3))*2) : (int → int) → (int →
    int)
```

OCamlの型 (3)

4. バリアント型、代数データ型 (参考)

```
type weekday = Monday | Tuesday | Wednesday | Thursday |
    Friday
Tuesday : weekday

type expr = CstI of int
          | Prim of string * expr * expr
CstI(3) : expr
Prim("+", CstI(3),CstI (4)) : expr
Prim("+", Prim("*",CstI(3),CstI(4)),CstI(5)) : expr
```

代数データ型は、バリアント型と再帰型を含む。
発展課題: GADT (一般化された代数データ型) とは何か調べよ。

型の例

整列関数その1(特定の型の要素からなるリストを整列):

```
Sort_int   :  int list   ->  int list
Sort_float :  float list  ->  float list
```

整列関数その2(任意の型のリストを整列):

```
Sort :  ∀α.  (α list  ->  α list)
```

整列関数その3(要素間の比較関数も引数とする):

```
Sort :  ∀α ((α * α -> bool) -> (α list  ->  α list))
Sort (<) [10; 30; 20] = [10; 20; 30]
Sort (>) [10; 30; 20] = [30; 20; 10]
```


実行時 vs コンパイル時

式 (1+"abc") が「いつ」エラーになるか。

- MiniC や MiniML では、実行時に (動的に) エラーになる。
- C や OCaml では、コンパイル時に (静的に) エラーになる。

エラーはなるべく早い段階で見つかる方がよい。

- 静的な検査の方が、(ディバッグの観点では) うれしい。

静的な型システム

式 (1+"abc") の整合性に関するエラーを、静的に発見したい。

- 式に「型」(type) を付け、その型を追うことによって整合性を検査する。
- 式の種類ごとに、どのような型が付くかを決めたものを「型システム」という。
- 型の整合性を検査するだけで、多くのバグを発見できる。
- 静的に検査ができていたら，実行時には検査は不要⇒実行時の効率が良くなる.

型システムの健全性 (Type Soundness):

- コンパイル時 (静的) に，型が整合したら，実行時の型の不整合 (実行時のエラー) は決して起きない。

型検査と型推論

型検査: 全ての変数 (や関数) の型が宣言されている言語で，型の整合性を検査すること．

- C 言語や Java 言語.
- 型検査は、変数や定数などのアトミックな式からはじめて、より大きな式の型が整合しているか検査する、という形式で行われる。

型推論: 変数 (や関数) の型が必ずしも宣言されていない言語で，その型を推論しつつ，型の整合性を検査すること．

- ML 言語や Haskell 言語.
- ML 言語では、「与えられた式に対して、最も一般的な型を推論する」という型推論アルゴリズムあり.

今日の課題

人間型推論器となり、以下の式の型 (あるいは「型がつかないこと」を導きなさい。

```
let f x = x + 3 in
let g y = f (f 10) in g (g 20)

let rec f x =
  if x = 0 then 91
  else if (x mod 2) = 0 then f (x/2)
  else if x = 100 then "Hi"
  else f (x*3 + 1)

let rec tarai x y z =
  if x <= y then y
  else tarai (tarai (x−1) y z)
             (tarai (y−1) z x)
             (tarai (z−1) x y)
```

発展課題: tarai はなにをするか？