色々な型

JAIST D2

宇田 拓馬

目次

Void型

- 値をもたない型 (というのが本来の意味)
- C言語の `void` などは実際にはただひとつの値をもつ (Unit型)
- → ボトム型, ユニット型

ボトム型

- 値をひとつも持たない型
- ゼロ型, 空型などとも
- カリー＝ハワード同型対応において偽に対応する型
- (構造的)部分型付けの言語ではすべての型のサブタイプ(部分型)となる
- 逆にすべての型のスーパータイプが `Any` (`Int` 型の値は `Any` 型の値でもある)
- Scalaでは `Nothing` がこの型に対応する
- 例外処理などで使われる(`throw` 式の型は `Nothing`)

```
def divide(a: Int, b: Int, accum: Int): Int = {
  if (b == 0) throw new IllegalArgumentException("zero divided")
  else if (a < b) accum
  else divide(a - b, b, accum + 1)
}
```

構造的部分型付け

- 構造が一致すれば同じ型だとみなせる
- `Cat` 型の値は `Animal` 型の値でもある

```
data Animal =
  { bark :: IO ()
  }

data Cat =
  { bark :: IO ()
  , punch :: IO ()
  }
```

ユニット型

- 値をただひとつだけもつ型
- カリー＝ハワード同型対応において真に対応する型
- ゼロ引数の関数の引数や, 副作用のある関数の返り値として用いられる
- 一般的な言語の `null` の型はすべての型のサブタイプでありただひとつの値 `null` をもつ型？

```
function hello(name) {
  console.log(`Hello, ${name}!`);
}

function getInput() {
  return document.getElementById("input").value;
}
```

```
hello :: String -> Effect Unit

getInput :: Unit -> Effect Element
```

幽霊型

- 型チェックのためだけの型
- 以下の論文で作られた造語
  - Leijen, Daan, and Erik Meijer. "Domain specific embedded compilers." ACM Sigplan Notices 35.1 (1999): 109-122.
- 実行時には存在しないためこんな名前
- 使いどころ
  - データの型は同じだが意味が異なる
  - 同じ型として扱いたい
- ちなみに構造的部分型付けの言語では使えない

```
data Plain
data Cipher
data Password a = Password String

encrypt :: Password Plain -> Password Cipher
```

直積型・直和型

- 直積型
  - 複数の型を組み合わせた型
  - カリー＝ハワード同型対応において論理積に対応する型
  - タプルやレコードで表されることが多い
- 直和型
  - 複数の型を列挙した型
  - カリー＝ハワード同型対応において論理和に対応する型
- 代数的データ型 (ADT: Algebraic Data Types)
  - 直積型と直和型によって構成される型

```
data List a
  = Nil
  | Cons a (List a)
```

依存型 (依存直積型)

- 値に依存した型
- カリー＝ハワード同型対応において全称量化に対応する型
- $$\forall x \in A. B(X)$$
- `長さ3の文字配列` とか(`Vect 3 Char`)
- $$\forall len \in Nat. Vect\ len\ elem$$

```
data Vect : (len : Nat) -> (elem : Type) -> Type where
  Nil : Vect Z elem
  (::) : (x : elem) -> (xs : Vect len elem) -> Vect (S len) elem
```

依存型 (依存直積型)

- `take` の型では引数の配列の長さが十分にあることが保証されている
- 例えば `take 3 [1, 2]` は型エラー

```
length : (xs : Vect len elem) -> Nat
length [] = 0
length (x::xs) = 1 + length xs

tail : Vect (S len) elem -> Vect len elem
tail (x:xs) = xs

take : (n : Nat) -> Vector (n + m) elem -> Vect n elem
take Z xs = []
take (S k) (x :: xs) = x :: take k xs
```

依存型 (依存直積型)

- では `filter` の型は？
- `(elem -> Bool) -> Vect len elem -> Vect ? elem`
- 結果の型の長さがわからない
- $\forall elem, len \in Nat. (elem -> Bool) -> Vect\ len\ elem -> (\exists p \in Nat. Vect\ p\ elem)$

依存ペア (依存直和型)

- ある値とその値に依存した型をもつ値のペア
- カリー＝ハワード同型対応において存在量化に対応する型
- $$\exists x \in A. (P(x))$$

依存ペア (依存直和型)

- Idrisで型 `3` の値が何になるのか知らない...

```
filter : (elem -> Bool) -> Vect len elem -> (p ** Vect p elem)
filter p [] = ( _ ** [] )
filter p (x::xs) =
  let (_ ** tail) = filter p xs
  in if p x then
    (_ ** x::tail)
  else
    (_ ** tail)
```

おまけ (カリー＝ハワード同型対応)

- プログラミング言語と証明論の対応関係
  - 型と命題
  - プログラムと証明
- 命題に対応した型について実装を書くとそれが証明になる
  → 定理証明支援系 (Coq, Isabelle, Agda, Idris, F*, Lean)

論理	プログラミング
全称量化 $\forall x \in A.\, B(x)$	依存直積 $\prod_{x:A} B(x)$
存在量化 $\exists x \in A.\, B(x)$	依存直和 $\sum_{x:A} B(x)$
含意 $A \supset B$	関数型 $A \to B$
論理積 $A \wedge B$	直積型 $A \times B$
論理和 $A \vee B$	直和型 $A + B$
真 $\top$	トップ型 1
偽 $\bot$	ボトム型 0

カリー＝ハワード同型対応[https://ja.wikipedia.org/wiki/カリー＝ハワード同型対応]