


プログラミング言語論 #15

Type Theory



2018-07-30



プログラミング言語の変化

| | C | C++ | JavaScript | Haskell |
|------|-----|-------------|------------|----------------------------|
| 1970 | K&R | | | |
| 1980 | | | | |
| 1990 | C99 | C++98 | 1.0 | Haskell 1.0, Haskell 98 |
| 2000 | | | | |
| 2010 | C11 | C+11, C++14 | 2.0 | Haskell 2010 型レベル, 種 |
| 2017 | | C++17 | | |
| 2018 | | | | Haskell2018 |

仕様が変わるのは常態

型推論の機能を持つ言語が増えてきた

型・型推論

Swiftでは値を代入する際に、静的な型推論を行っています。そのため、型宣言を明示的に記述することなく、自動的に型を決定できます。次のコードを見てください。

```
var count = 10
count = "This is String" // コンパイルエラー
```

型推論とは、プログラミング言語で用いる変数や関数の型をリテラルやその他コンテキストの情報によって自動的に決定する機構のことです。Objective-Cのような明示的な型の宣言は、Swiftでは必須でなくなりました。

Haskell を始めとする関数型言語に影響を受けた言語も出てきた

変数・定数

Swiftには、変数・定数を宣言する方法が2つあります（「//」以降はコメントです）。

```
var count = 10 // 変数の宣言
let name = "John" // 定数の宣言

count = 12
name = "Sam" // コンパイルエラー
```

varを用いて宣言した変数は、何回でも代入することを許容します。変数の型が同じであれば、何回でも代入を繰り返すことができます。

一方、**let**を用いて宣言した定数は、再代入を許しません。letを用いると、状態、副作用を極力排する方向にコードを記述できます。

Go and Rust

Go

静的型付, ad-hoc 多相, 例外処理

Rust

強い静的型付, 型クラス, ad-hoc 多相, 型推論

型の意義

- 安全性の確保：実行時エラーの可能性を減らす
- 高速化：型情報を用いた高速な命令列への変換

私見

- 「プログラム＝計算する」：論外
- プログラミング：記号的構造記述
- 大規模プログラミング：その延長が必要

型研究例：型レベル計算

計算の一部を型のレベルに持ち上げコンパイル時に実行する
実行時にはエラーがない

- スタック
- 違う型が混ざっても構わないリスト
- 通信に失敗しないサーバクライアント通信

型による安全性の実現

no warranty

```
1  -- | スタックから最初の要素を取り除く
2  pop :: Stack x → Stack x
3  pop s = 何も考えずに最初の要素を取り除く, そして実行時エラー
```

run-time error detection

```
1  pop :: Stack x → Stack x
2  pop s = if 空でない then 最初の要素を取り除く else エラー
```

compile-time error detection

```
1  -- 自然数の定義を利用する (※これはまだ依存型ではない)
2  data Stack x a = Stack x Zero | Stack x (Succ n)
3  pop :: Stack x (Succ n) → Stack x n  -- 空ではない宣言
4  pop s = 空ではないので何も考えずに要素を取り除いてエラーなし
```


型研究例：依存型

依存型; dependent type

ある引数の値に応じて別の部分（引数や返値）の型が決まる

依存関数（**返値の型**が引数の**値**に依存する）：

- 引数が 0 なら double を返す
- 引数が正整数なら配列を返す

例 2：

- 引数が 0 なら push メソッドを持つスタック型 S1 を返す
- 引数が正整数なら push メソッドと pop メソッドを持つスタック型 S2 を返す
- push メソッドは S*型を受け取り S2 型を返す
- pop メソッドは S2 型を受け取り状況に応じて S1 または S2 型を返す

型研究例：篩型

- 指定された型の中の指定された値しか引数に取らない
- 指定された型の中の指定された値しか返値として返さない

型に関する計算について

様々な型機能を導入した言語では型推論の停止性は保証されない

例：（ある条件下において）これらは型推論（正規化された型付け）できない¹

```
1 omega = (λx. x x)(λy. y y)           -- 関数omega p.278
2 T = ∀X <: Top. ¬(Y <: X.¬Y)         -- データ型T p.337
```

言語に条件（制約）を加えることで型推論は可能になる

プログラミング言語（の意味論）は注意深く設計されている

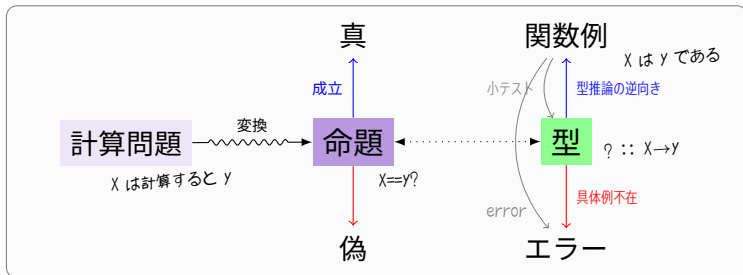
Hindley-Milner アルゴリズム

（関数型言語に対して用いられる）代表的型推論アルゴリズム。
第6回で紹介したのはそのイントロ（制限された多相まで対応）。

もちろん、それらは制約が加えられた言語になっている。

¹ Benjamin C. Pierce, 型システム入門 プログラミング言語と型の理論, オーム社, 2013(原本 2002).

命題と型の対応

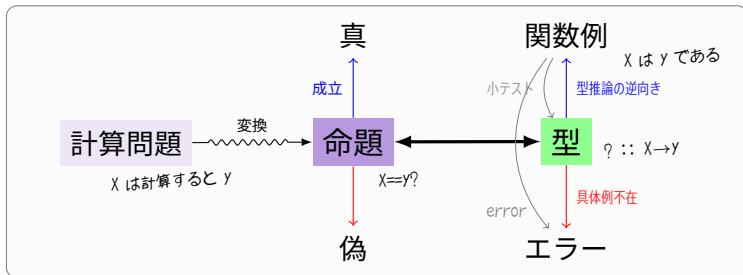


- 関数の型を決める問題はある論理式（命題）を解く問題に対応 いって場合によっては停止しない
- 逆に、命題の真偽を決める問題はある型の関数を見つける問題に対応

両者は 1 対 1 に対応する (草稿 1969, 論文 1980)

すなわち、ある命題を証明するには対応する型を持つ関数が定義できることを（実例で）示せばよい。

命題と型の対応



- 関数の型を決める問題はある論理式（命題）を解く問題に対応 いって場合によっては停止しない
- 逆に、命題の真偽を決める問題はある型の関数を見つける問題に対応

両者は 1 対 1 に対応する (草稿 1969, 論文 1980)

Curry-Howard 同型対応; Curry-Howard Isomorphism

命題と型は同型である。

すなわち、ある命題を証明するには対応する型を持つ関数が定義できることを（実例で）示せばよい。

定理証明支援系 Coq

Coq

[▶ web](#)

定理証明支援系システム． Coq はコンパイラに対応する． コンパイルが通る＝証明終了．

証明するプログラムを書き、実行するのではなく、コンパイルエラー（型エラー）がない時点で証明が完成

Coq などの定理証明支援系による成果

- 4 色問題を証明
- Kepler 予想の証明: <http://wired.jp/2014/09/21/sphere-packing/>
- Feit-Thompson 定理（奇数位数の有限群は可解群）の査読
- 暗号システム OpenSSL が情報を漏らさないことを証明

<https://x80.org/rhino-coq/>

- JavaScript への移植版（開発中）
- デモは任意の整数 m よりも大きな素数が存在することの証明

以下を証明せよ

$$a + b = b + a$$

$$a \times b = b \times a$$

深さ n の2分木は深さ $n - 1$ の2分木を少なくとも1つ含む

第6回の型推論は有限時間で停止する

このプログラムはセグメント違反を起こさない