# Coq 入門

# 定理証明支援系とは

- 証明が正しいことを検証してくれる装置。
- 公理から定理の証明を構築する手順を、型システムで模倣する。
- （証明は人間が作る）
- 検証だけでなく、証明の構築も支援してくれる。

定理証明支援系の１つとして **Coq** がある。

# インストール

Coq 8.15 まではインストーラが用意されている。最新は 8.16

- **新しさより簡単さ重視する人**：レポジトリからインストーラを入手
- **最新版を入れたい人**：Coq/SSReflect/MathCompの設定に従う
  - `sudo apt get install` でなく `sudo apt install`
  - WSLでは「新しいライブラリの設定が要るかも」の箇所で `sudo apt install -y libgmp-dev` が必要だった(参考)

# エディタ

- **VScode+VSCoq** がおすすめ。VScode上で拡張機能(`Ctrl+Shift+X`)から「vscoq」と検索すれば出てくる。
  - Windowsの人はWSL上で動かせるように **Remote - WSL** も入れよう（「wsl」で検索）。
- **CoqIDE** もインストーラから入れた場合は利用できる（がVSCoqの方が便利）
- **Emacs 上で Proof General を動かす手段**もあるらしい。
- **jsCoq**：ブラウザ上で動くCoq。

# まずは簡単な例。

```
Section ModusPonens.
Variables P Q : Prop.
Theorem MP : P -> (P -> Q) -> Q.
Proof.
  intros Pis PimpliesQ.
  apply PimpliesQ.
  apply Pis.
Qed.
```

VSCoq, jsCoq上では `Alt+↓` で１文ずつ証明が読み込まれる。

ここで `Print MP.` を実行すると

```
MP =
fun (Pis : P) (PimpliesQ : P -> Q) =>
PimpliesQ Pis
     : P -> (P -> Q) -> Q

Arguments MP _ _%function_scope
```

のように表示される。

つまり `MP` とは、引数として `Pis`(値)と `PimpliesQ`(関数)を受け取って `PimpliesQ Pis` を返す関数。

Coqの(正確には**Gallina**の)書式は `変数名 : 型` 。

`MP` の型 `P -> (P -> Q) -> Q` は、 `P` 型の値と `P->Q` 型の値(関数)を受け取って `Q` 型の値を返す関数、を意味する。

これを、「Pという言明の証明とP->Qという言明の証明を受け取ってQという言明の証明を作る」ことと同一視する。

→ **Curry-Howard同型対応**

（はじめから

```
Definition MP := fun (x : P) (y : P -> Q) => y x.
```

としても同じ。証明が長くなってくるとそれは困難。）

# Curry-Howard同型対応における解釈

| 論理演算 | 解釈 |
| --- | --- |
| $A \wedge B$ | $A$の証明と$B$の証明のペアの集合（直積集合） |
| $A \vee B$ | $A$の証明の集合と$B$の証明の集合の和集合 |
| $A \to B$ | $A$の証明からの$B$の証明の構成方法の集合 |
| $\forall x : A, B(x)$ | $A$の証明$x$からの$B(x)$の構成方法の集合（依存型） |
| $\exists x : A, B(x)$ | $A$の証明$x$と$B(x)$のペアの集合（依存型） |

依存型は、証明の型が$x$に依存する。

# 帰納的に定義される型

`Print bool.` を行うと

```
Inductive bool : Set :=  true : bool | false : bool.
```

と表示される。 `true` と `false` が `bool` 型を構成していてその他にはない、と分かる。

自然数も帰納的に定義されている。`Print nat.` を行うと

```
Inductive nat : Set :=  O : nat | S : nat -> nat.
```

- `O`（これは0のこと）は `nat` 型で
- `nat` 型の値に `S` を作用させたものも `nat` 型で
- それらだけが `nat` 型である

ことが分かる。

`S` を作用させることは次に大きい自然数を作ることに対応している。
`O` ↔ 0, `S O` ↔ 1, `S (S O)` ↔ 2, `S (S (S O))` ↔ 3……。

命題としての`True`，`False`（`bool`型の`true`，`false`とは異なる）や`and`，`or`も帰納的に定義されている。

# SSReflect

- Coq には標準ライブラリに加えて，SSReflect，MathComp という強力なライブラリがある．（この2つは現在は同じライブラリとしてまとめられている．）
- このライブラリを読み込むことで，定理だけでなく新しいタクティックまで導入することができる．

# move

- `move=>->` : `intro TMP; rewrite TMP; clear TMP`。 `move->` とも書ける。

# Coq の効用

- とても長大で，全てを人間が検査することが難しい証明の正しさを保証できる．（例．四色定理，奇数位数定理）
- ソフトウェアにバグがないことを保証できる．（例：CompCert）

# リンク集

- Coq クィックリファレンス
  事例集をみればとりあえず証明が書ける。
- TopProver
  競プロのCoq版。最近はコンテストは開かれていないが過去問で証明の練習ができる。
- Coq/SSReflect/MathComp Tutorial
  スライドとかExample がいろいろ載ってる。
- ソフトウェアの基礎
  網羅的に勉強できそう。

# リンク集2

- Coq/SSReflect/MathCompの文献

# 参考文献


- 萩原学/アフェルト・レナルド『Coq/SSReflect/MathComp による定理証明』(森北出版，2018)　必携。
- 定理証明支援系 Coq による形式検証（PDF）