# 付録 A　Coq システムを使った演習

2013 年度までの本講義では、自前の CAL システム[*20]を使った演習を行ってきた。このシステムは現在でも利用可能ではあるが、いくぶん古くなってきたため、2014 年度から、汎用の定理証明支援システムである Coq を利用し、この上に、命題論理や型付きラムダ計算の体系を構築して演習を行なうこととした。

これらのシステムを利用する目的は、形式的体系のチェックやアルゴリズムで処理できる部分を、ソフトウェアに委ねることにより、演習効果を高めることである。特に、紙と鉛筆の演習では、学生の解答が正しいかどうかを人手で判定する必要があるが、これらのソフトウェアを使うことにより，自分の好きな時に，好きなペースで，演習を行うことができるようになる．

## A.1　Coq システムに必要な環境

プログラムを書く場合に、プログラム言語 (の処理系) 以外に、プログラムを記述/編集/実行/ディバッグ等をやりやすくするための IDE と呼ばれるシステムを利用することが多い。

それと同様に、Coq システムを利用する場合にも IDE を利用するのがよい (IDE なしで Coq の証明を書くことはできるが、開発効率が相当に悪くなる)。Coq では、ProofGeneral と CoqIDE と呼ばれる 2 つの IDE があるが、ここでは、ProofGeneral を使うこととする。(ProofGeneral 自体は、“General” という名前があらわしている通り、Coq に限らず、いろいろな定理証明支援系の IDE となることのできる汎用のシステムである。)

ProofGeneral は emacs の上で動作するので、まとめると、Coq + ProofGeneral + emacs というソフトウェアの上で Coq を使うことになる。(なお、本演習の解答を、CoqIDE を用いて作成することは可能であるので、自分で CoqIDE の解説を読んでそちらを使ってもよい。)

## A.2　準備

Coq+ProofGeneral システムを使うための準備は、ごく簡単であり、自分の .emacs ファイル (ホームディレクトリ直下に置かれる) に ProofGeneral を利用するための設定を追加するだけである。この記述は、たとえば、以下のようなものである。

```
(load "/opt/local/share/ProofGeneral/generic/proof-site.el")
(defadvice coq-mode-config (after deactivate-holes-mode () activate)
  "Deactivate holes-mode when coq-mode is activated."
  (progn (holes-mode 0))
)
(add-hook 'proof-mode-hook
   '(lambda ()
      (define-key proof-mode-map (kbd "C-c C-j") 'proof-goto-point)))
```

[*20] Computation and Logic, 京都大学の佐藤雅彦教授が中心となり亀山らが協力して構築してきた教育用ソフトウェア.

なお、上記の記述は、ProofGeneral が置かれた場所によって若干異なるので、演習の際の手引き (ウェブページ) を参考にされたい。

2014 年 10 月現在で、coins システムには、Coq version 8.4pl4 がインストールされている。(自分のノート PC 上に Coq 等を載せて演習をやりたい人は、同じバージョンを載せるようにしてほしい。)

## A.3 命題論理の世界 (ProgLogic)

Coq の中に命題論理の世界を構築し、命題論理における証明の演習を行えるようにした。Coq 自身は命題論理を包含しているが、Coq が持っている命題論理の機能は、この授業では使わず、それとは別に命題論理の世界を構築していることに注意されたい。このため、この演習での論理記号や証明のための命令は、Coq におけるそれらとは異なっている。

論理式について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| $\bot$ | `False` | 「矛盾」を意味する論理式 |
| $P, Q, R$ | `P, Q, R` | 原子命題 (基本命題) |
| $A \wedge B$ | `A /\ B` | 「かつ」 |
| $A \vee B$ | `A \/ B` | 「または」 |
| $A \supset B$ | `A -> B` | 「ならば」 |
| $\neg A$ | `~A` | 「でない」 |
| $A, B, C \vdash B$ | `[A;B;C] \|- P` | 判断 (judgment) |

表 3　論理式の表記の対応

次に、命題論理の証明で使う推論規則の対応を述べる。

| 講義資料での規則 | 演習システムでの規則 | 説明 |
|---|---|---|
| assume | assume. | 仮定をする規則 |
| $\supset I$ | `impI.` | 「ならば」導入規則 |
| $\supset E$ | `impE (A).` | 「ならば」除去規則、$A \supset B$ の $A$ を指定する |
| $\wedge I$ | `andI.` | 「かつ」導入規則 |
| $\wedge EL$ | `andEL (B).` | 「かつ」除去 (左) 規則、$A \wedge B$ の $B$ を指定する。 |
| $\wedge ER$ | `andER (A).` | 「かつ」除去 (右) 規則、$A \wedge B$ の $A$ を指定する。 |
| $\vee IL$ | `orIL.` | 「または」導入 (左) 規則。 |
| $\vee IR$ | `orIR.` | 「または」導入 (右) 規則 。 |
| $\vee E$ | `orE (A) (B).` | 「または」除去規則、$A \vee B$ の A と B を指定する。。 |
| $\neg I$ | `notI.` | 「でない」導入規則。 |
| $\neg E$ | `notE (A).` | 「でない」除去規則、$\neg A$ の A を指定する。 |
| $\bot E$ | `falseE.` | 「矛盾」除去規則 |
| $\neg\neg E$ | `notnotE.` | 二重否定除去規則 (古典論理でのみ使える) |

表 4　推論規則の表記の対応

なお、今回のシステムでは、日本語文字 (全角文字) は一切使っていない。たとえば、「ならば」の記号は、->という 2 文字であらわすのであって、「→」という文字は使わない。

## A.4 単純型付きラムダ計算の世界 (SimpleType)

Coq の中に、単純型付きラムダ計算 (ただ、直積型と関数型を持つもの) を構築した。

型について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| $\bot$ | `Empty` | 空の型 |
| $S, T, U$ | `S, T, U` | 型変数 (あるいは型定数) |
| $A \times B$ | `A * B` | 直積型 |
| $A + B$ | `A + B` | 直和型 |
| $A \to B$ | `A -> B` | 関数型 |
| $\neg A$ | `A -> Empty` | 関数と空の型で記述できる |
| $M : T$ | `M \in T` | $M$ は $T$ 型 |
| $x : A, y : B \vdash M : U$ | `[x \in A; y \in B] \|- M \in U` | 判断 (judgment) |

表 5　型の表記の対応

次に、項について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| $(M, N)$ (または $\langle M, N \rangle$) | `(M,N)` | 対 |
| $left(M)$ | `left(M)` | 左への射影 |
| $right(M)$ | `right(M)` | 右への射影 |
| $inl(M)$ | `inl(M)` | 直和への埋め込み |
| $inr(M)$ | `inr(M)` | 直和への埋め込み |
| $case(x, (y : B)M, (z : C)N)$ | `case(x,(y \in B)M,(z \in C)N)` | 場合分け |
| $\lambda(x : A)M$ | `\lambda (x:A) M` | 関数 (ラムダ式) |
| $M@N$ (または $M\ N$) | `M @ N` | 関数の適用 (使用) |
| $abort(M)$ | `abort(M)` | 実行の中断 |

表 6　項の対応

最後に、型付け規則について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

規則の名前にはすべて大文字の R がついていて、Rvar のようなっていることに注意せよ。これは、Case や left という Coq の命令があるため、それらの重複を避けるためである。

演習システム (Coq の中に作った SimpleType の世界) は、parser については十分こなれていないため、ラムダ式を入力したとき、予想外の括弧付けをしてしまうことがある。今回の演習では、括弧を多目につけて対処してほしい。

上記のほか、SimpleType.v の後ろの方の問題では、解くべき問題に `exists_term t,` と記載され、ラムダ式の部分が `t` という変数になっている。これは、その部分に適切なラムダ式をいれる、という部分も解答者

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| var | `Rvar` | 変数導入 |
| pair | `Rpair` | 対 |
| left | `Rleft (S)` | $T \times S$ の $T$ への射影 |
| right | `Rright (T)` | $T \times S$ の $S$ への射影 |
| inl | `Rinl` | 直和への埋め込み (左から) |
| inr | `Rinr` | 直和への埋め込み (右から) |
| case | `Rcase` | 場合分け |
| apply | `Rapply (T)` | 関数適用,$T \to S$ の $T$ を指定 |
| lambda | `Rlambda` | ラムダ式 |
| abort | `Rabort` | 異常終了 |

表 7　型付け規則の対応

が考えなければいけない。たとえば、以下のセッションは、この種の問題に対する解答である。

```
Theorem ex133 : exists_term t, [] |- t \in T -> T.
Proof.
  Ex (\lambda (x \in T) x).
  Rlambda. Rvar.
Qed.
```

証明 1 行目の Ex ではじまる部分で、t の具体形を入力している。なお、演習システムでは、ラムダ式の中の変数は、英字の小文字 1 文字に限定している。(a や x は変数になるが、a1 や x134 は変数ではない。)

## A.5　関数型プログラム言語の体系 (CoreML)

関数型プログラム言語の体系 (CoreML) についても型付けの演習問題を用意した。ただし、こちらは、Coq のプリミティブなキーワードとぶつかるものがあり、「講義 (あるいは本資料) のキーワード」と「演習システムにおけるキーワード」とが、大幅にちがっているので注意してほしい。

型と判断について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| `int` | `int` | 整数型 |
| `bool` | `bool` | 真理値型 |
| $A \times B$ | `A * B` | 直積型 |
| $A \to B$ | `A -> B` | 関数型 |
| $M : T$ | `M \in T` | $M$ は $T$ 型 |
| $x : A, y : B \vdash M : U$ | `[x \in A; y \in B] \|- M \in U` | 判断 (judgment) |

表 8　型と判断の表記の対応

項について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| 10 | `(% 10)` | 整数定数 |
| $-13$ | `(% -13)` | 整数定数 |
| `true` | `_true` | 真理値定数 |
| `false` | `_false` | 真理値定数 |
| $M = N$ | `M = N` | 比較 (等しさ) |
| $M > N$ | `M > N` | 比較 (大なり) |
| $(M, N)$ | `(M , N)` | 対 |
| `left`$(M)$ | `_left M` | 左への射影 |
| `right`$(M)$ | `_right M` | 右への射影 |
| $\lambda x.M$ | `_lambda (x) M` | 関数 (ラムダ式) |
| $M@N$ | `M @ N` | 関数の適用 (使用) |
| `if` $L$ `then` $M$ `else` $N$ | `_if L then M else N` | 条件式 |
| `let` $x = M$ `in` $N$ | `_let x = M in N` | let 式 |
| `fix` $f.x.N$ | `_fix f (x) M` | 再帰関数 |

表 9　項の対応

ほとんどのキーワードの先頭に underscore (_ という記号のこと) がついていることに注意してほしい。なお、`_lambda` は長いので `_lam` と書いてもよい。ラムダ式と再帰関数において、引数をあらわす `(x)` の括弧は省略できない。

型付け規則について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

| 講義資料での表現 | 演習システムでの表現 | 説明 |
|---|---|---|
| var | `Rvar` | 変数 |
| int | `Rint` | 整数定数変数 |
| bool | `Rbool` | 真理値定数 |
| plus | `Rplus` | 加算 |
| eq | `Req` | 比較 (等しさ) |
| comp | `Rcomp` | 比較 (大なり) |
| pair | `Rpair` | 対 |
| left | `Rleft (S)` | $T \times S$ から $T$ への射影 |
| right | `Rright (T)` | $T \times S$ から $S$ への射影 |
| lambda | `Rlambda` | ラムダ式 |
| apply | `Rapply (T)` | 関数適用、$T \to S$ の $T$ を指定 |
| if | `Rif` | 条件式 |
| let | `Rlet (S)` | let 式、束縛変数の型を指定 |
| fix | `Rfix` | 再帰関数 |

表 10　型付け規則の対応

規則の名前にはすべて大文字の R がついていて、Rvar のようなっていることに注意せよ。

演習システム (Coq の中に作った SimpleType の世界) は、parser については十分こなれていないため、ラムダ式などを入力したとき、予想外の括弧付けをしてしまうことがある。今回の演習では、括弧を多目につけて対処してほしい。