Jacques Garrigue, 2019 年 4 月 24 日

述語論理とSSReflectのタクティク

1　述語論理

前回見た命題論理は，推論の概念を捉えているが，具体的な対象に対して議論することができない．我々が一般的に使う論理はその拡張である述語論理になる．最も見慣れているのは一階述語論理だが，実数の形式化などには二階述語論理が必要．

論理式　二階述語論理の論理式は以下の結合子から定義される．(X を除くと一階述語論理に限定される)

t	$::=$	x	項変数	P,Q	$::=$	$\ldots$	命題
	$\mid$	c	項定数		$\mid$	ϕ	命題述語
	$\mid$	$f(t_1,\ldots,t_n)$	項関数		$\mid$	$\phi(t_1,\ldots,t_n)$	述語
ϕ	$::=$	p	述語名		$\mid$	$\forall v.P$	全称
	$\mid$	X	述語変数		$\mid$	$\exists v.P$	存在
v	$::=$	$x \mid X$	変数		$\mid$	$t_1 = t_2$	等価性
σ	$::=$	$t \mid \phi$	代入				

導出規則　命題論理の導出規則に以下の規則を加える．

$\forall$ **導入**　$\dfrac{\Delta \vdash P \quad v \notin fv(\Delta)}{\Delta \vdash \forall v.P}$

$\forall$ **除去**　$\dfrac{\Delta \vdash \forall x.P}{\Delta \vdash [t/x]P} \quad \dfrac{\Delta \vdash \forall X.P}{\Delta \vdash [\phi/X]P}$

反射率　$\Delta \vdash t = t$

$\exists$ **導入**　$\dfrac{\Delta \vdash [t/x]P}{\Delta \vdash \exists x.P} \quad \dfrac{\Delta \vdash [\phi/X]P}{\Delta \vdash \exists X.P}$

$\exists$ **除去**　$\dfrac{\Delta \vdash \exists v.P \quad \Delta, P \vdash Q \quad v \notin fv(\Delta, Q)}{\Delta \vdash Q}$

代入　$\dfrac{\Delta \vdash [t_1/x]P \quad \Delta \vdash t_1 = t_2}{\Delta \vdash [t_2/x]P}$

ここで自由変数 $fv(P)$ と代入が利用される．

$fv(x) = \{x\}$　　$fv(c) = \emptyset$　　$fv(f(t_1,\ldots,t_n)) = \bigcup fv(t_i)$

$fv(True) = fv(False) = fv(p) = \emptyset$　　$fv(\phi(t_1,\ldots,t_n)) = fv(\phi) \cup \bigcup fv(t_i)$

$fv(P \supset Q) = fv(P \wedge Q) = fv(P \vee Q) = fv(P) \cup fv(Q)$　　$fv(X) = \{X\}$

$fv(\forall x.P) = fv(\exists x.P) = fv(P) \setminus \{x\}$

$[t/x]x = t$　　$[t/x]y = y$　　$[\phi/X]t = t$

$[t/x]c = c$　　$[t/x]f(t_1,\ldots,t_n) = f([t/x]t_1,\ldots,[t/x]t_n)$

$[\sigma/v]\,True = True$　　$[\sigma/v]False = False$　　$[\phi/X]X = X$　　$[\sigma/v]X = X$

$[\sigma/v](P \supset Q) = [\sigma/v]P \supset [\sigma/v]Q$　　$\wedge, \vee$も同様

$[\sigma/v](\phi(t_1,\ldots,t_n)) = [\sigma/v]\phi([\sigma/v]t_1,\ldots,[\sigma/v]t_n)$

$[\sigma/v]\forall w.P = \forall w.[\sigma/v]P \quad (w \notin fv(\sigma) \cup \{v\})$　　$[\sigma/v]\forall v.P = \forall v.P$

$[\sigma/v]\exists w.P = \exists w.[\sigma/v]P \quad (w \notin fv(\sigma) \cup \{v\})$　　$[\sigma/v]\exists v.P = \exists v.P$

導出の例

$$\dfrac{\dfrac{\forall x.human(x) \supset mortal(x) \vdash \forall x.human(x) \supset mortal(x)}{\forall x.human(x) \supset mortal(x) \vdash human(S) \supset mortal(S)} \qquad human(S) \vdash human(S)}{\forall x.human(x) \supset mortal(x), human(S) \vdash mortal(S)}$$

2 Coq との対応

Coq では項と述語が型をもっており，代入が型を保たなければならない．さらに，型自身が型 (ソート) をもっている．

$$\vdash t, x : T \text{ かつ } \vdash T : Set \text{ または } \vdash T : Type$$

$$\vdash P : Prop$$

$$\vdash p, X : T_1 \to \ldots \to T_n \to Prop$$

さらに，全ての $T \to P$ が $\forall x : T, P$ の略記法であり，$\forall$ と $\to$ に関する規則が統一されている．

抽象 $\dfrac{\Delta, v : S \vdash t : T}{\Delta \vdash (\mathtt{fun}\ v{:}S \Rightarrow t) : \forall v{:}S, T}$ **適用** $\dfrac{\Delta \vdash t : \forall v{:}S, T \quad \Delta \vdash u : S}{\Delta \vdash (t\ u) : [u/v]T}$

$\Delta \vdash t : T$ が項 t が仮定 Δ の元で型 T をもつという意味である．T が命題なら ($\vdash T : Prop$)，t はその証明項．今までの導出規則が拡張され，$\vdash$ と結論の間に証明項が含まれる．ただし，抽象が $\to$ の導入と $\forall$ の導入を兼ね，適用が $\to$ の除去と $\forall$ の除去と述語および項関数の構成を兼ねる．たとえば，公理の規則は $\Delta \vdash x : T \ \ (x : T \in \Delta)$ になる．

上の導出の例は適用 2 回でできる．

```
Section Socrates.
Variable A : Set.
Variables human mortal : A -> Prop.
Variable socrates : A.

Hypothesis hm : forall x, human x -> mortal x.
Hypothesis hs : human socrates.

Theorem ms : mortal socrates.
Proof.
  apply: (hm socrates).
  assumption.
Qed.

Print ms.                                          (* 定義を表示する *)
ms = hm socrates hs                              (* ((hm socrates) hs) *)
     : mortal socrates
End Socrates.
```

3 全称と存在

$\forall$ と $\exists$ の間に De Morgan の法則がなりたつ．前回と同様に，$\exists$ を導出しようとしたときに classic を使わなければならない．

```
Section Laws.
Variables (A:Set) (P Q : A->Prop).

Lemma DeMorgan2 : (~ exists x, P x) -> forall x, ~ P x.
Proof.
  move=> N x Px. elim: N. by exists x.
Qed.

Theorem exists_or :
  (exists x, P x \/ Q x) -> (exists x, P x) \/ (exists x, Q x).
Proof.
  move=> [x [Px | Qx]]; [left|right]; by exists x.
Qed.

Hypothesis classic : forall P, ~ ~ P -> P.

Lemma DeMorgan2' : (~ forall x, P x) -> exists x, ~ P x.
Proof.
  move=> nap.
  apply: classic => nenp.
  apply: nap => a.
  apply: classic => npa.
  apply: nenp.
  by exists a.
Qed.

End Laws.
```

練習問題 3.1 以下の定理を *Coq* で証明せよ.

```
Section Coq3.
  Variable A : Set.
  Variable R : A -> A -> Prop.
  Variables P Q : A -> Prop.

  Theorem exists_postpone :
    (exists x, forall y, R x y) -> (forall y, exists x, R x y).
  Theorem or_exists : (exists x, P x) \/ (exists x, Q x) -> exists x, P x \/ Q x.

  Hypothesis classic : forall P, ~ ~ P -> P.
  Theorem remove_c : forall a,
    (forall x y, Q x -> Q y) ->
    (forall c, ((exists x, P x) -> P c) -> Q c) -> Q a.
End Coq3.
```

4 System F

Coq の *Prop* 型は Girard が考案した System F という型体系を実装しており, $\forall$ と適用のみで二階述語論理を表現できる.

```
Section SystemF.
Definition Fand P Q := forall X : Prop, (P -> Q -> X) -> X.
Definition For P Q := forall X : Prop, (P -> X) -> (Q -> X) -> X.
Definition Ffalse := forall X : Prop, X.
Definition Ftrue := forall X : Prop, (X -> X).
Definition Feq T (x y : T) := forall P, P x <-> P y.
Definition Fex T (P : T -> Prop) := forall X : Prop, (forall x, P x -> X) -> X.
```

```
Theorem Fand_ok (P Q : Prop) : Fand P Q <-> P /\ Q.
Proof.
  split => [pq | [p q] X].
  + split; by apply: pq.
  + by apply.
Qed.

Theorem For_ok (P Q : Prop) : For P Q <-> P \/ Q. Abort.
Theorem Ffalse_ok : Ffalse <-> False. Abort.
Theorem Ftrue_ok : Ftrue <-> True. Abort.
Theorem Feq_ok T (x y : T) : Feq x y <-> x = y. Abort.
Theorem Fex_ok T (P : T -> Prop) : Fex P <-> exists x, P x. Abort.
End SystemF.
```

各論理演算子の System F での表現がその除去規則を模倣している．

特に等価性に関する定義 `Feq` は Leibniz equality と言い，古くから知られている．x と y が等しいとは，任意の述語 p について，$p(x)$ と $p(y)$ が同値であること，言い換えれば，x と y を区別する述語が存在しないこと．

練習問題 4.1 *System F* での符号化に関する定理を証明せよ．

5 タクティク・タクティカル・修飾子

前回はタクティクと論理規則との関係について述べたが，SSReflect のタクティクが多くの機能を秘めており，ここでその一部を説明する．

基本タクティク

`move`, `apply`, `done`, `case`, `split`, `left`, `right`, `elim`, `have`, `suff`, `rewrite`, `set` でほぼの証明ができるが, 特に `move` と `rewrite` が修飾子を多く使う．

Coq の証明状態を以下とする．(ただし焦点がない場合もある)

仮定 Δ
==========
焦点 P -> 結論 Q

各タクティクがゴールの各部 $(\Delta; P \vdash Q)$ を論理的規則に基づいて変えていく．その変化を説明する．

`move` 修飾子なしでは何もしないが，修飾子によって仮定と前提の移動や左規則の適用が可能．

`apply` ゴールを適用した定理の前提に変える．規則 Apply-R を参照．
前の状態: $\Delta; P_1 \to \ldots \to P_n \vdash Q$
後の状態: $\Delta; \vdash P_i$ $(1 \leq i \leq n)$ に置き換わる

`done` 様々な自明な解決法を試み，解決できなければエラーを起こす．公理，矛盾，反射率を含む．

`case` 焦点に対して場合分けを行う．単独では `move=> []` とほぼ同じ．規則 $\wedge$-L と $\vee$-L を参照．

`split` $\wedge$-R を参照．ゴールを二つに分割する．

`left`, `right` $\vee$-R を参照．ゴールの左か右を選ぶ．間違えると証明ができない可能性が大きい．

`elim` 焦点の返り値 (結論) に対して場合分けを行う．型によって帰納法の原理が適用される．併せて，`apply` と同様に焦点の前提もゴールのリストに加えられる．焦点が否定ならば
前の状態: $\Delta; \neg P \vdash Q$
後の状態: $\Delta; \vdash P$

have $H : P$

新しい仮定 $H : P$ を証明した上で現在のゴールの証明を続ける．
前の状態: $\Delta;\vdash Q$
後の状態: $\Delta;\vdash P$ と $\Delta, H : P;\vdash Q$
仮定名 H が省略されると代わりに $\Delta; P \vdash Q$ になる．

suff $H : P$

have と同じだが，ゴールの順番が逆になる．新しい仮定の元で現在のゴールを証明した後にその仮定を証明しなければならない．have $H : P$; last first とほぼ同じ．

rewrite */def*

ゴールの焦点および結論の中で定義 *def* を展開する．詳しい rewrite の修飾子は後で述べる．

rewrite *lemma*

補題 *lemma* の結論が等式 $t_1 = t_2$ ならば，ゴールの焦点および結論の中で t_1 を t_2 に書き換える．等式の変数が自動的に選ばれる．*lemma* の前提がゴールリストに追加される．

set $x := t$

仮定に定義 $x := t : T$ を加える．x は仮定名, t は項, T は t の型．同時に結論の中に t を x に置き換える (rewrite -/x と同じ)．
t の中に穴「_」を含めてもいい．その場合，結論の中に当て嵌る項を探し，その穴を自動的に埋める．

have $H := t$

H は仮定名，t は項．Δ の元で t の型が T なら，仮定 $H : T$ を置く．t が証明項なら，T が命題になる．

基本タクティカル　タクティクの前後に書き，そのタクティクの動作を拡張する．

: move, apply, case, elim の直後に使える．タクティク本体を実行する前に前提を置く．
前の状態: $\Delta;\vdash Q$
タクティカル: : $H_1 \ldots H_n$
後の状態: $\Delta'; P_1 \vdash P_2 \to \ldots \to P_n \to Q$
$H_i : P_i$ が Δ に含まれる仮定なら，Δ' から除かれる．そうでなければ，H_i は Δ の元で片付け可能な項で，P_i はその型である．なお $H_i : P_i$ が Δ に含まれても，(H_i) と書くことで置いたままにできる．また，不要になった仮定 K があれば，$\{K\}$ と書くことで消すことができる．

=> 全てのタクティクの後に使える．後に書かれた名前が順番に前提を焦点から仮定に変える．

[] は焦点に左規則や分解規則を適用する．中に名前や修飾子を書いてもいい．[l_1 | ...| l_n] の形で分解後の各場合に対する列も書ける．

- move 以外のタクティクと組合せたとき，=>の後の最初の [] か [l_1 | ...| l_n] が生成された各ゴールに対するものになり，分解を行わない．その前に-を書くと分解になる．

/= は焦点と結論を単純化する．

// は全てのゴールに対して done で解決を失敗せずに試みる．

//= はその両方を行う．

/H は項 H を焦点に適用する．

/(_ H) は焦点を H 適用する．

-> 焦点が等式 $t_1 = t_2$ ならば t_1 を t_2 に書き換え，その後に焦点が消される．

<- は逆向きに書き換える.

$\{K\}$ は「:」のときと同じ意味.

/H apply の直後に使うと H が結論に適用される．move や case の直後だと，タクティクを実行する前に H が焦点に適用される.

in $H_1 \ldots H_n$ ほぼ： $H_1 \ldots H_n$ => $H_1 \ldots H_n$ と同じ意味だが，操作が元の結論に適用されない.

tac$_1$; *tac*$_2$ *tac*$_1$ を実行した後に，生成された全てのゴールに対して *tac*$_2$ を実行する.

tac ; [*tac*$_1$ | ...| *tac*$_n$]
tac が n 個のゴールを生成した場合，それぞれに対して $tac_1, \ldots, tac_n$ を実行する.

by *tactic*
tactic ; done と同じ．なお，*tactic* が「;」を含んでもいい．(次の「.」まで行く)

by [] done と同じ.

do n *tactic* *tactic* を n 回繰り返す.

do !*tactic* *tactic* を可能な限り繰り返す.

rewrite

rewrite は独自の構文を使う．基本的には，定理の名前や適用を並べる.

```
rewrite lem1 lem2 (lem3 n 1)
```

各定理に対して，繰り返しや適用箇所の指定もできる.

!*lem* 定理 *lem* による書き換えを可能な限り繰り返す.
n!*lem* 定理 *lem* による書き換えを n 回繰り返す.
?*lem* 定理 *lem* を 0 または一回使う.
-*lem* 定理 *lem* を左向きに使う.
$\{n\}$*lem* n 番目の出現を書き換える
[p] *lem* パターン p(穴のある項) にマッチする最初の出現を書き換える.
/def 定義 *def* を展開する.
-*/def* 定義 *def* を畳み込む.
(*lem*$_1$,...,*lem*$_n$)
lem$_1$... *lem*$_n$ を順番に試して，最初に成功してものを使う．他の修飾子 (特に「!」) と組合せてもいい.
(_ : t_1 = t_2)
t_1 を t_2 に置き換える． t_1 = t_2 がゴールリストに追加される.

上記の書き換え修飾子を組合せることができるが，順番に気を付けなければならない.

```
rewrite -{2}[_ + n]lem
```

また，定理や定義の間に評価修飾子 (/=, //, //=) を挿入してもいい.