Coq による定理証明入門 *

髙橋　真

2023 年 4 月 26 日

この資料は定理証明支援系 Coq を利用して定理証明の初歩を分かりやすく解説する[*1]ことを目的としています.

* この講義資料は JSPS 科研費 26560089 の助成を受けた研究の過程で得られたものです.

[*1] ただし，プログラミング言語としての特徴については扱わないので，Coq の全体像については参考文献「ソフトウェアの基礎」[1] を参照してほしい.

目次

2017 年 11 月 4 日作成
2017 年 11 月 10 日修正
2017 年 11 月 15 日修正（集合族と同値関係の追加）
2020 年 8 月 11 日修正（タクティクスと証明の見直しによる修正）
2020 年 8 月 13 日修正（追加の修正）
2021 年 5 月 24 日修正（フォント指定のミスの修正）
2023 年 4 月 26 日修正（誤字の修正）

1 Coq

1.1 Coq とは

Coq はフランスの INRIA で開発されている定理証明支援系である．Coq の公式ページには以下のように記載されている．

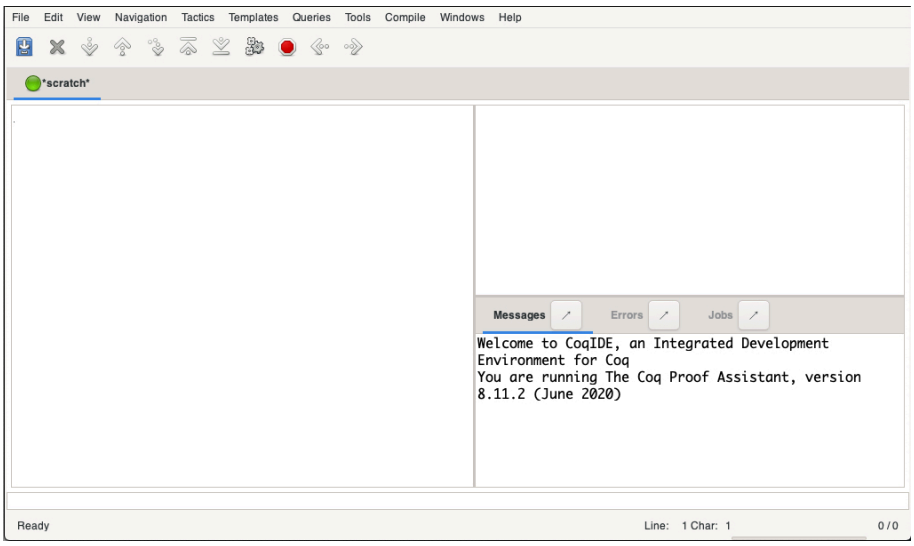
> Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. Typical applications include the certification of properties of programming languages (e.g. the CompCert compiler certification project, or the Bedrock verified low-level programming library), the formalization of mathematics (e.g. the full formalization of the Feit-Thompson theorem or homotopy type theory) and teaching.

1.2 Coq のインストール

Coq は `https://github.com/coq/coq/releases` よりダウンロードできる．Windows OS や Mac OS X の場合はバイナリをダウンロードする．Linux 系の OS の場合は，パッケージ管理ツールを利用してインストールする．Coq のライセンスは GNU LGPL ライセンスである．

1.3 CoqIDE のウインドウ

Coq はコマンドラインでも実行可能であるが，ここでは CoqIDE を用いて説明する．CoqIDE を起動すると 3 ペインのウインドウが開く．



左側が編集で，右上は仮定とゴールの表示，右下は情報を表示する．

1.4 基本的な枠組み

Coq を使用する場合，基本的なライブラリーは読み込まれているが，必要に応じて外部ライブラリーを読み込むことができる（2.1.9 参照）．古典論理のライブラリーを読み込まない限り，Coq で証明できるのは直観主義論理で証明できるもののみになる．`Section` 宣言は使わなくてもよいが，局所的な変数を使う場合は必要になる．`Section` を宣言する場合は，

`Section` ⟨ セクション名 ⟩.

のようにセクション名の後にピリオドをつける．`Section` の終わりは，

`End` ⟨ セクション名 ⟩.

とする．また，書いた内容を反映させるためには，式の評価を行なう必要がある．単純な式の評価はメニューバーの下矢印をクリックする．あるいは Navigation メニュで Forward を選択してもよい．戻るにはメニューの上向き矢印（BackWard）を使う．他に全てを一度に評価したり (End)，先頭に戻したり (Start)，指定された場所まで評価を行う (Go to）ことができる．キーボードショートカットを使うのが便利である．キーボードショートカットで使用する Modifier キーは，Edit->Preferences->Shortcuts で変更できる．

2 論理式の証明

2.1 命題論理

Coq では，すべての変数は型をもっている．例えば，`Prop` は命題の型を表していて，`A:Prop` のように表し，変数 `A` が型 `Prop` をもつことを表す．

論理式を証明するときは，証明する論理式の前に，`Goal`，`Theorem`，`Lemma` などを書く．証明した論理式を後で参照する必要のない場合は `Goal` でよいが，あとでその事実を証明に利用するのであれば，`Theorem` や `Lemma` にする．`Theorem` と `Lemma` に違いはなく，どちらにするかは気持ちの問題である．証明する論理式には名前を付ける[*2]．そのセクションで共通に使う変数は，`Variable` で宣言する．

```
Variable A B C:Prop.
```

命題 A, B に対し，$A \to (B \to A)$ が成り立つことを証明する場合，

```
Lemma HA1:A->(B->A).
```

とする．論理結合子の $\to$ は Coq では，ハイフン-と大小記号>で表す．`HA1` が論理式 `A->(B->A)` につけた名前[*3]である．これらを評価すると Coq は proof-editing モードに入り，右上のペインに仮定とゴールが現れる．

[*2] とりあえず定理証明する際に参照するための区別と考えればよいが，詳細は参考文献 [1], [2] を参照してほしい．

[*3] この論理式は，ヒルベルトによる形式式的体系の公理の一つなので HA1 と名前をつけている．

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
```

```
1 subgoal
A, B, C : Prop
______________________________________(1/1)
A -> B -> A
```

証明を始める場合，Proof. を付けるが，付けなくてもよい．証明を終えたら（Coq が No more subgoals. となったら）Qed. で終える．これで上記であれば，HA1 として，A->(B->A) が利用できる．

2.1.1 ゴールが含意の場合

実際に A->(B->A) を証明してみる．LJ の推論規則[*4]の 右→ は，

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \to B} \quad (\text{右}\to)$$

であるから，$\vdash A \to (B \to A)$ を示すには，$A \vdash B \to A$ を示せば良い．このように A を仮定に入れて，ゴールを $B \to A$ に変更するには，intro タクティック[*5]を用いる．intro. として評価するとそのように仮定とゴールが変換されることがわかる．

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro.
```

```
1 subgoal
A, B, C : Prop
H : A
______________________________________(1/1)
B -> A
```

もう一度，intro を行うと，$A, B \vdash A$ となり，ゴールと同じ仮定ができる．なお，評価したタクティックの次は空白を入れる．入れないと先頭が赤字で表示され次の評価ができない．

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro.intro.
```

```
1 subgoal
A, B, C : Prop
H : A
______________________________________(1/1)
B -> A
```

ゴールと同じ仮定がある場合，LJ では始式から構造に関する推論規則で導かれるが，Coq では trivial あるいは assumption を用いる．trivial. を評価すると No more subgoals. となり証明が終了する．

[*4] LJ あるいは LK については，参考文献 [3] を参照してほしい．Γ を論理式の列とするとき，$\Gamma \vdash X$ の直観的な意味は Γ を仮定すると X が導くことができることである．

[*5] タクティックについては [4] を参照してほしい。

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro. intro. trivial.
```

```
No more subgoals.
```

最後に Qed. で Lemma HA1 が登録される.

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro. intro. trivial.
Qed.
```

上記では，intro を 2 回続けたが，これは intros で 1 回で行うことができる．ただし，intros は単なる intro の繰り返しではない．2.1.6 で説明する．なお，intro を行うと Coq が自動的に仮定に入る論理式に名前をつけるが，H，H0 のように機械的な番号付になるので，自分で名前を付けたければ，intro K. のように指定できる．

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro K.
```

```
1 subgoal
A, B, C : Prop
K : A
______________________________________(1/1)
B -> A
```

仮定が多く出てくる場合は，このようにして名前を制御した方がよいが，このテキストでは記載を簡単にするため一部を除いて特に指定はしないで証明を進める．

今挙げた例のように Coq では，$\Gamma \vdash A \to B$ を証明するには，$A, \Gamma \vdash B$ を証明すればよいというように，LJ の推論規則（あるいは LJ で許容される推論規則）を元にタクティックを用いてゴール変換を行いながら推論を進める．なお，基本はゴール変換による証明であるが，仮定を書き換えた方が良い場合もあるので，適宜ゴール変換と仮定の書換を交えて証明をすすめる．

以下，いくつかの例を用いながら，タクティックの説明をする．タクティックの名前を忘れた場合は，メニューの Tactics を見て探してほしい

2.1.2 ゴールが論理積のとき

Lemma Iand:A->(B->A/\B).

を証明しようとして，intros を行うとゴールが論理積 A/\B になる．なお，論理結合子の $\wedge$ は Coq では，スラッシュ/とバックスラッシュ\[*6]で表す．

[*6] 日本語キーボードでは\は￥キーをタイプする．

```
Lemma Iand:A->(B->A/\B).
Proof.
intros.
```

```
1 subgoal
A, B, C : Prop
H : A
H0 : B
_____________________________________(1/1)
A /\ B
```

ゴールが論理積 $X \wedge Y$ の形のときは，split でゴールを X と Y の 2 つのゴールに分割しそれぞれ証明する．これは，推論規則でいうと

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

に対応している．今の例では，split で分割後，trivial で証明が終了する．

```
Lemma Iand:A->(B->A/\B).
Proof.
intros. split.
trivial. trivial.
Qed.
```

注意：ここでは説明のために intros してから，split したが，split の定義は intros をした後に apply conj を適用するものなので，最初の intros は不要でいきなり split を適用してもよい．

2.1.3 仮定に論理積があるとき

Lemma And1:A/\B -> A

を証明しようとして，intro を行うと，仮定に論理積 A/\B が入る．

```
Lemma And1:A/\B->A.
Proof.
intro.
```

```
1 subgoal
A, B, C : Prop
H : A /\ B
_____________________________________(1/1)
A
```

仮定が論理積 $X \wedge Y$ の形のときは，destruct あるいは inversion で仮定を分割する．仮定にタクティックを適用する場合は，どの仮定に適用するか指定する．ここでの使い方での destruct と inversion の違いは，destruct が元の論理積が仮定に残らないのに対し，inversion は元の論理積を仮定に残したままにすることである．

```
Lemma And1:A/\B->A.
Proof.
intro. destruct H.
```

```
1 subgoal
A, B, C : Prop
H : A
H0 : B
_____________________________________(1/1)
A
```

```
Lemma And1:A/\B->A.
Proof.
intro. inversion H.
```

```
1 subgoal
A, B, C : Prop
H : A /\ B
H0 : A
H1 : B
_____________________________________(1/1)
A
```

いずれにせよ，仮定にゴールと同じ論理式があるので，`trivial` で証明ができる．

```
Lemma And1:A/\B->A.
Proof.
intro. destruct H.
trivial.
Qed.
```

これは推論規則として考えると，

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

に対応している．

2.1.4 ゴールが論理和のとき

`Lemma Or1:A-> A\/ B`

を証明しようとして，`intro` を行うと，ゴールが論理和 `A\/B` になる．論理結合子の論理和 $\vee$ は Coq ではバックスラッシュとスラッシュである．

```
Lemma Or1:A->A\/B.
Proof.
intro.
```

```
1 subgoal
A, B, C : Prop
H : A
_____________________________________(1/1)
A \/ B
```

ゴールが論理和 $X \vee Y$ のときは，LJ では背理法が使えないので，証明できる論理式が X であるか Y であるかを `left`, `right` で指定する必要がある．ここでは，`A` が仮定にあるので，`left` を選択する．

```
Lemma Or1:A->A\/B.
Proof.
intro. left. trivial.
Qed.
```

推論規則としては，

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

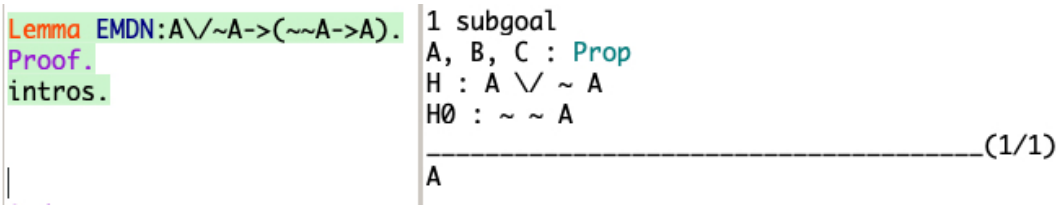
に対応している．背理法でないと証明できない場合については，2.1.9 で扱う．

注意：split のときと同様に，left はそもそも intros の後に apply or_introl を適用するので，最初の intro は不要で，最初から left で証明できる．

2.1.5 仮定に論理和があるとき

```
Lemma EMDN:A\/~A->(~~A -> A)
```

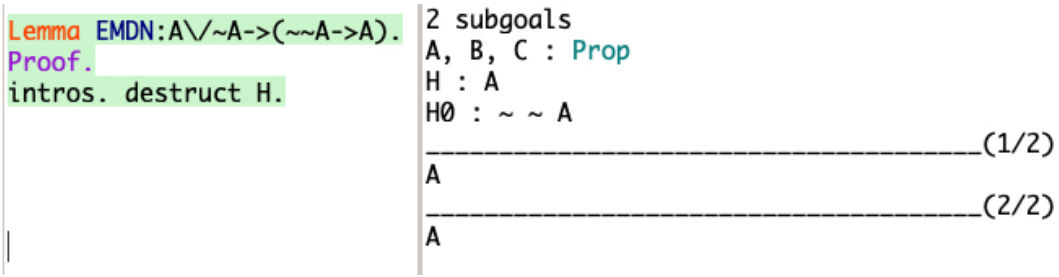
を証明しようとして，intro を行うと，仮定に論理和 A\/~A が入る．論理結合子の否定 ¬ は，Coq ではチルダ~で表す．



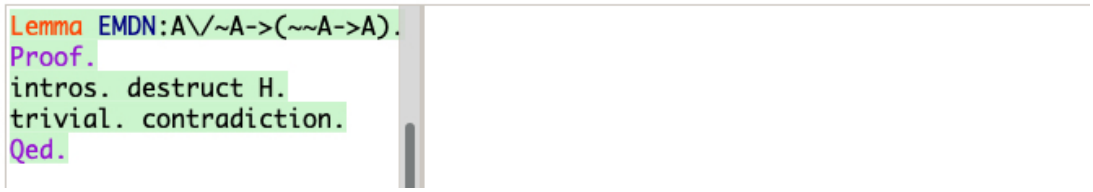
仮定が論理和 $X \vee Y$ の形のときは，destruct あるいは inversion で仮定を分離し，X の場合でも Y の場合でもいずれの場合でも証明できることを示す．推論規則では

$$\frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}$$

に対応している．



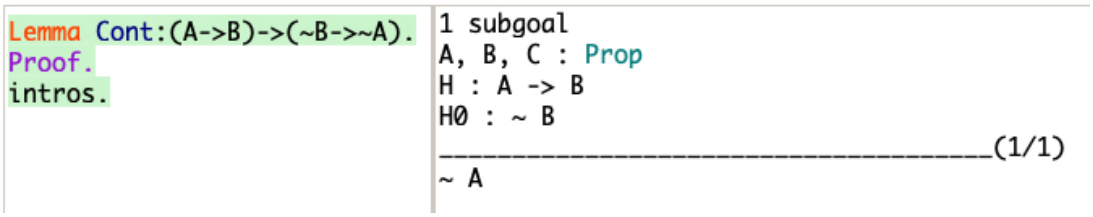
この例では，A\/ ~A を destruct あるいは inversion することでゴールが 2 つでき，A を仮定する場合は trivial，~A を仮定する場合は~~A が仮定にあり仮定が矛盾するので，contradiction で証明できる．



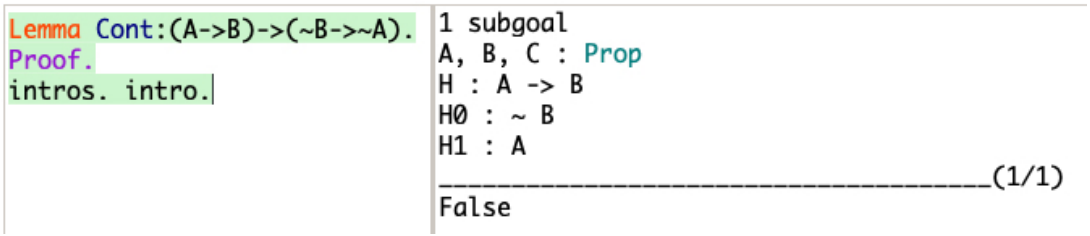
2.1.6　ゴールが否定形のとき

```
Lemma Cont:(A->B)->(~B -> ~ A)
```

を証明しようとして，intros を行うと，ゴールが否定形~A になる．



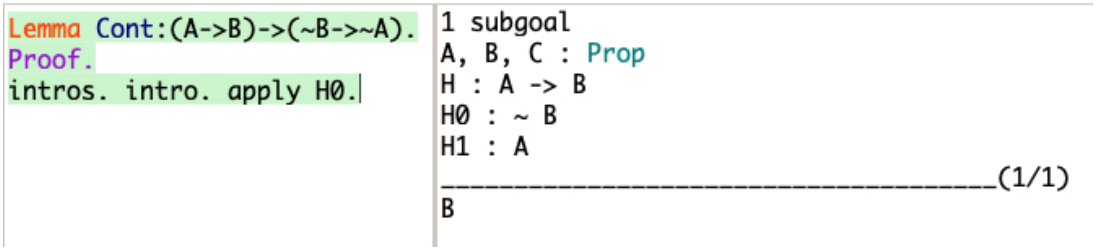
Coq では，$\neg A$ は $A \to \bot$（Coq では $\bot$ は False）の省略系である．従って，intro を使うことができる．



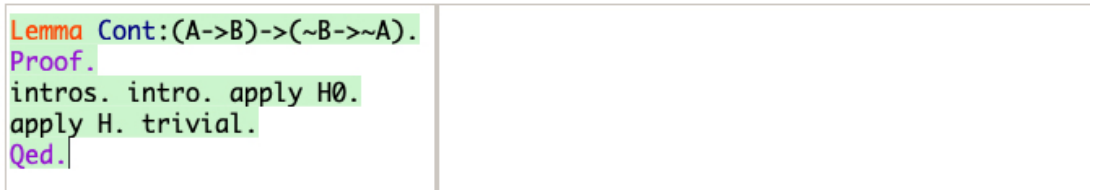
最初に適用した intros では，~A には適用されていなくて，明示的に intro を使う[*7]ことで，A を仮定に入れることができている．仮定の~B は B-> False であるから，False を示すには，B を示せばよいことがわかる．推論規則としては，

$$\frac{\Gamma \vdash X \qquad \Gamma, Y \vdash Y}{\Gamma, X \to Y \vdash Y}$$

に対応している．（上式右は明らかに成り立つので，上式左のみを示せばよい．）これは apply タクティックを使う．



再度，apply タクティックを使えば証明が終了する．

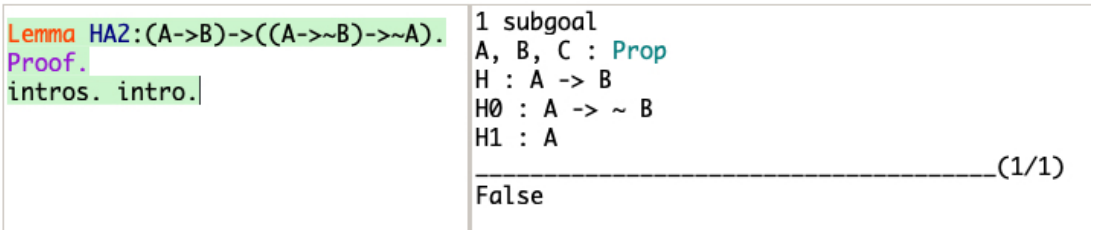


[*7] このことからわかるように intros は intro の単なる繰り返しではない．

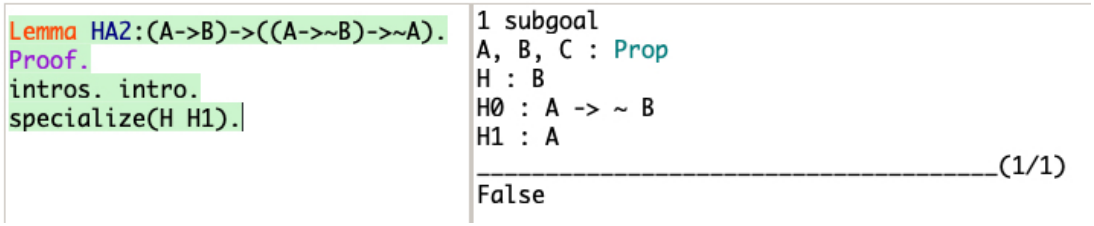
2.1.7　仮定に X と $X \to Y$ があるとき

Lemma HA2:(A->B)->((A->~B)->~A)

を証明しようとして，intros を行い，続けて intro を行うと，仮定に A，A->B，A->~B が入る．

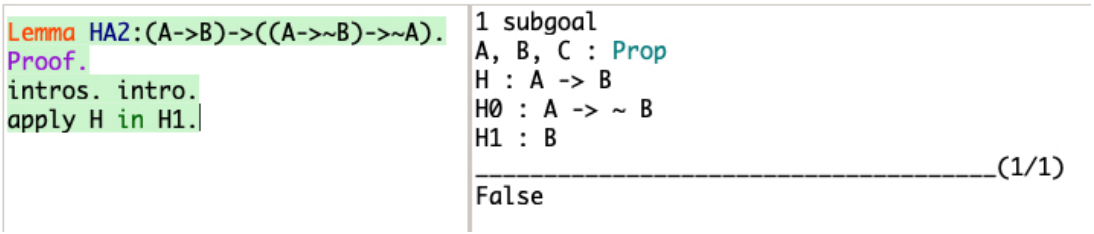


仮定に X と $X \to Y$ があるとき，仮定を Y にすることができる．X を Y に書き換える場合と $X \to Y$ を Y に書き換える場合でタクティックが異なる．specialize を用いて specialize(H H1) とすると，仮定 H:A->B が B に変わる．



H:A->B を A から B への写像と考えて，H1:A の H1 を A の元だと考えると，H(H1) は B の元になるが，この H(H1) を (H H1) と表して，それを実際に計算させるのが specialize になっていると考えられる．

specialize ではなく，apply H in H1 とすると，仮定 H1:A が B に変わる．



今の場合，仮定 H1:A を書き換えると，論理式 A が消えて H0:A->~B への適用ができなくなる[*8]ため，前者の specialize を用いることにする．specialize を H と H0 に続けて適用すると，仮定に B と~B が入り矛盾するので証明が終わる．

[*8] 後で説明する generalize と intro を使用して A のコピーを残しておけば apply でも証明はできる．

```
Lemma HA2:(A->B)->((A->~B)->~A).
Proof.
intros. intro.
specialize(H H1).
specialize(H0 H1).
contradiction.
Qed.
```

注意：この例題では仮定を書き換える説明のために specialize を使った証明をしたが，absurd を用いる方が自然である．absurd は LJ で許容される推論規則

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}$$

に対応している．absurd B. を適用すると，B と~B の 2 つのゴールが生成され，仮定を apply することで証明が終わる．

```
Lemma HA2:(A->B)->((A->~B)->~A).
Proof.
intros. intro.
absurd B.
```

```
2 subgoals
A, B, C : Prop
H : A -> B
H0 : A -> ~ B
H1 : A
_____________________________________(1/2)
~ B
_____________________________________(2/2)
B
```

```
Lemma HA2:(A->B)->((A->~B)->~A).
Proof.
intros. intro.
absurd B.
apply H0. trivial.
apply H. trivial.
Qed.
```

2.1.8　同値式のとき

$A \leftrightarrow B$ は $(A \to B) \wedge (B \to A)$ の略記であるから、それに準じてタクティックを適用する．例えば、ゴールにある場合は split、仮定にある場合は destruct あるいは inversion を使う．

演習問題 1. 次の論理式が証明可能であることを Coq で示せ．

1. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
2. $\neg(A \wedge B) \to (A \to \neg B)$
3. $((A \wedge B) \to C) \to (A \to (B \to C))$
4. $(B \to A) \to ((C \to A) \to ((B \vee C) \to A))$
5. $(\neg A \wedge \neg B \leftrightarrow \neg(A \vee B)$
6. $(\neg A \vee \neg B) \to \neg(A \wedge B)$

2.1.9　古典論理

ここまでは排中律 $A \vee \neg A$ を必要としない論理式を証明してきた．Coq では，特に仮定を置かない限り，直観主義論理 LJ で証明可能なものしか証明できない．排中律も証明することはできない．実際，排中律を証明しようとしても，ゴールが論理和の形であるから，`left` か `right` しかタクティックが使えない．従って，排中律を仮定するしか，古典論理は扱えない．

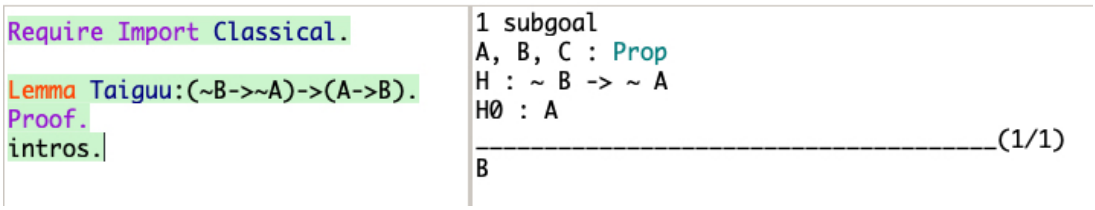
古典論理を扱うために必要／便利な公理・定理を集めたライブラリーが Classical である．ライブラリーを読み込むには次のように宣言する[*9]．

```
Require Import Classical.
```

このライブラリーでは，次のように公理，Lemma が扱われている．

```
Axiom classic : forall P:Prop, P \/ ~ P.
Lemma NNPP : forall p:Prop, ~ ~ p -> p.
Lemma Peirce : forall P:Prop, ((P -> False) -> P) -> P.
Lemma not_imply_elim : forall P Q:Prop, ~ (P -> Q) -> P.
Lemma not_imply_elim2 : forall P Q:Prop, ~ (P -> Q) -> ~ Q.
Lemma imply_to_or : forall P Q:Prop, (P -> Q) -> ~ P \/ Q.
Lemma imply_to_and : forall P Q:Prop, ~ (P -> Q) -> P /\ ~ Q.
Lemma or_to_imply : forall P Q:Prop, ~ P \/ Q -> P -> Q.
Lemma not_and_or : forall P Q:Prop, ~ (P /\ Q) -> ~ P \/ ~ Q.
Lemma or_not_and : forall P Q:Prop, ~ P \/ ~ Q -> ~ (P /\ Q).
Lemma not_or_and : forall P Q:Prop, ~ (P \/ Q) -> ~ P /\ ~ Q.
Lemma and_not_or : forall P Q:Prop, ~ P /\ ~ Q -> ~ (P \/ Q).
Lemma imply_and_or : forall P Q:Prop, (P -> Q) -> P \/ Q -> Q.
Lemma imply_and_or2 : forall P Q R:Prop, (P -> Q) -> P \/ R -> Q \/ R.
```

従って，このライブラリーを読み込むとこれらの Lemma を利用して証明することができる．例えば，`(~B-> ~A)->(A->B)` は古典論理でないと証明できない論理式であるが，これを証明してみよう．`intros` を適用するとゴールが B になるが，このままでは適用できる仮定がない．



そこで，Classical の Lemma NNPP をゴールに適用する．`apply NNPP.` とすると Coq が今のゴール B に合わせて NNPP を適用する．

[*9] Section 内でライブラリーを読み込むことは現在は推奨されていない．必要なライブラリーは Section 開始前に宣言する．ここでは説明の都合上途中で宣言している．

```
Require Import Classical.

Lemma Taiguu:(~B->~A)->(A->B).
Proof.
intros. apply NNPP.
```

```
1 subgoal
A, B, C : Prop
H : ~ B -> ~ A
H0 : A
______________________________________(1/1)
~ ~ B
```

ゴールが~~B になったので intro が適用でき，2.1.6 で使用した specialize または absurd を用いて証明ができる．

```
Lemma Taiguu:(~B->~A)->(A->B).
Proof.
intros. apply NNPP.
intro. specialize(H H1).
contradiction.
Qed.
```

あるいは，

$$\frac{\Gamma \vdash \neg A}{A, \Gamma \vdash \bot}$$

は LJ で許容される推論規則だが，これに対応するタクティックである generalize を使うこともできる．

```
Lemma Taiguu:(~B->~A)->(A->B).
Proof.
intros. apply NNPP.
intro. generalize H0.
```

```
1 subgoal
A, B, C : Prop
H : ~ B -> ~ A
H0 : A
H1 : ~ B
______________________________________(1/1)
A -> False
```

A->False は~A のことなので，後は仮定の H を apply すれば証明できる．

```
Lemma Taiguu:(~B->~A)->(A->B).
Proof.
intros. apply NNPP.
intro. generalize H0.
apply H. trivial.
Qed.
```

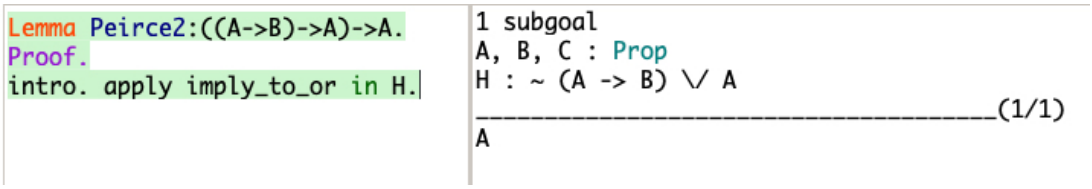
ライブラリーの Lemma を使うもう一つの例として Peirce の論理式[*10]$((A \to B) \to A) \to A$ を証明してみよう．

```
Lemma Peirce2:((A->B)->A)->A.
Proof.
intro.
```

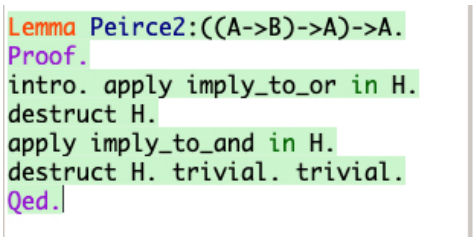
```
1 subgoal
A, B, C : Prop
H : (A -> B) -> A
______________________________________(1/1)
A
```

[*10] Classical の中にも Peirce という名前の Lemma があるが，それはこの論理式の特別の形で通常こちらを Peirce の論理式とよぶ．

intro をすると，仮定が (A->B)->A で結論が A になる．すぐに仮定を適用してゴールを A->B にすると，証明できる形にはならない．古典論理でないと証明できない論理式なので Classical の公理か Lemma の適用が必要である．証明の方法はいろいろあるが，ここでは，imply_to_or:forall P Q : Prop, (P -> Q) -> ~ P \/ Q．を仮定 H に利用してみる．タクティックの書き方は apply imply_to_or in H. である．



仮定が論理和に変わったので，destruct で分割する．仮定~(A->B) は，imply_to_and が使えそうである．



演習問題 2. 次の論理式が古典論理で証明可能であることを Coq で示せ．

1. $\neg(A \wedge B) \to \neg A \vee \neg B$
2. $((A \to \neg A) \to B) \to ((A \to B) \to B)$
3. $(\neg B \to \neg A) \to ((\neg B \to A) \to B$
4. $(A \to B) \vee (B \to A)$　　（ヒント：最初に NNPP を使う．）

最後に End PropositionalLogic. で Section を閉じる．Section から抜けると、Section 内で変数として仮定されていたものが forall の形で Section 内の定義に自動的に付加される．次にメニューの File->Save でファイルを保存する．拡張子は.v にする．Section を閉じると、Section 内で Variable として仮定されていたものに対して，その Section 外で，任意の論理式にあてはめて使用することができる．例えば，次のように Section PropositionalLogic を閉じた後に，次のような Lemma を示すことができる．

```
Section PropositionalLogic.
Variable A B C:Prop.

Lemma HA1:A->(B->A).
Proof.
intro. intro. trivial.
Qed.

End PropositionalLogic.

Section Another.

Variable X Y Z:Prop.

Lemma HAXY:(X->Y)->(Z->(X->Y)).
Proof.
apply (HA1 (X->Y) Z).
Qed.
```

apply (HA1 (X->Y) Z) は，Section PropositionalLogic で定義した HA1 の A として X->Y，B として Z を当てはめて適用することを示している．ただし，これは同一ファイル内で行なっているのでそのまま使えているが，他のファイルの場合はコンパイルしてから，その.vo ファイルを読み込む必要がある．詳しくは 3.2 で説明する．

2.2 述語論理

述語論理を Coq で扱うには，述語を定義する必要がある．述語は対象領域の要素が与えられると真偽が定まる命題になることを踏まえ，次のように宣言する．

```
Section PredicateLogic.
Variable A:Type.
Variable P Q:A->Prop.
Variable R:A->(A->Prop).
Variable t:A.
```

ここで、A は対象領域を表しているが、どのような型に対しても適用できるように Type 型にしる．また、P,Q は A 上の 1 変数述語、R は A 上の 2 変数述語、t は A の要素としている．

2.2.1 仮定が全称論理式のとき

Lemma all_imply:(forall x:A, P x)-> P t

を証明しようとして，intro を行うと，仮定に全称論理式 forall x:A, P x が入る．

```
Lemma all_imply:(forall x:A,P x)-> P t.
Proof.
intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : forall x : A, P x
______________________________________(1/1)
P t
```

仮定が全称論理式の場合、任意の項に対して成り立つことが仮定できる．推論規則としては

$$\frac{\Gamma, P[t/x] \vdash C}{\Gamma, \forall x P \vdash C}$$

に対応している．この例の場合は仮定 H を t に対して適用するため、apply (H t) とする．

```
Lemma all_imply:(forall x:A,P x)-> P t.
Proof.
intro. apply (H t).
Qed.
```

2.2.2 ゴールが存在論理式のとき

Lemma imply_exists:P t->(exists x:A,P x)

を証明しようとして，intro を行うと，ゴールが存在論理式 exists x:A,P x になる．

```
Lemma imply_exists:(P t)->exists x,P x.
Proof.
intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : P t
______________________________________(1/1)
exists x : A, P x
```

ゴールが存在論理式の場合、古典論理でない限り、存在論理式をみたす項 t（witness とよぶ）を明示しなければならない．この例の場合は、witness として t を選択すればよいので、exists t として t を適用する．

```
Lemma imply_exists:(P t)->exists x,P x.
Proof.
intro. exists t. trivial.
Qed.
```

2.2.3 ゴールが全称論理式のとき

Lemma alpha_all:(forall x:A, P x)-> forall y:A,P y

を証明しようとして，intro を行うと，ゴールが全称論理式 forall y:A, P y になる．

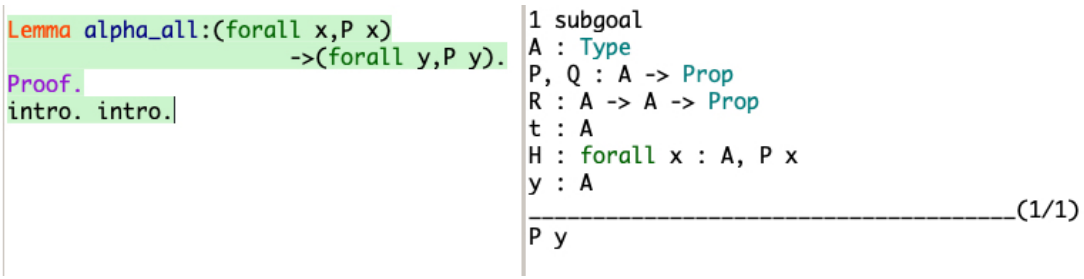
```
Lemma alpha_all:(forall x,P x)
                        ->(forall y,P y).
Proof.
intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : forall x : A, P x
______________________________________(1/1)
forall y : A, P y
```

ゴールが全称論理式の場合は、intro を行う．推論規則としては

$$\frac{\Gamma \vdash P[a/t]}{\Gamma, \vdash \forall x P} \qquad \text{（ただし、}a\text{ は下式には現れない変数）}$$

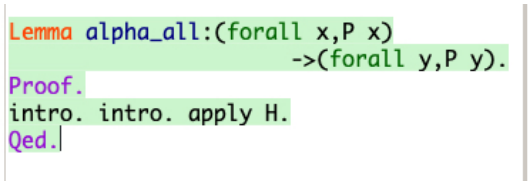
に対応する．この推論規則では自由変数 a に変数条件があるが，Coq では intro を適用すると、自動的にそれまでに自由変数として使われていない新しい変数が用意される． 今の場合 y は自由変数として使われていないので y がそのまま使われる[*11]．



```
Lemma alpha_all:(forall x,P x)
                      ->(forall y,P y).
Proof.
intro. intro.
```

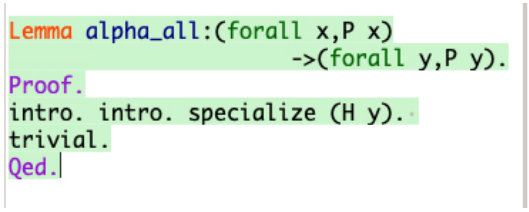
```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : forall x : A, P x
y : A
______________________________________(1/1)
P y
```

後は、仮定 H を apply を適用して証明が終わる．



```
Lemma alpha_all:(forall x,P x)
                      ->(forall y,P y).
Proof.
intro. intro. apply H.
Qed.
```

最後のところで使用した apply H. の代わりに，specialize (H y) と仮定を書き換えても証明できる．



```
Lemma alpha_all:(forall x,P x)
                      ->(forall y,P y).
Proof.
intro. intro. specialize (H y).
trivial.
Qed.
```

ここで証明した $\forall x P(x) \to \forall y P(y)$ は，束縛変数を書き換えてもよいことを表す．このように束縛変数を書き換えることを α 変換とよぶ．

2.2.4 仮定が存在論理式のとき

Lemma all_not_not_ex:(forall x:A, ~P x)-> ~exists x:A,P x

を証明しようとして，intro を続けて行うと，仮定に存在論理式 exists x:A, P x が入る．

[*11] 言語で変数記号を自由変数と束縛変数に分ける場合もあるが，Coq では分けていないので注意する．

```
Lemma all_not_not_ex:(forall x:A, ~P x)
      -> ~exists x:A,P x.
Proof.
intro. intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : forall x : A, ~ P x
H0 : exists x : A, P x
______________________________________(1/1)
False
```

仮定が存在論理式の場合は destruct を行う．推論規則としては

$$\frac{\Gamma, P[a/x] \vdash C}{\Gamma, \exists x P \vdash C} \qquad \text{（ただし、}a\text{ は下式には現れない）}$$

に対応する．Coq では destruct を適用すると、自動的にそれまでに自由変数として使われていない新しい変数が用意される．この例の場合 x は自由変数として使われていないので x がそのまま使われる．

```
Lemma all_not_not_ex:(forall x:A, ~P x)
      -> ~exists x:A,P x.
Proof.
intro. intro. destruct H0.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : forall x : A, ~ P x
x : A
H0 : P x
______________________________________(1/1)
False
```

後は仮定 H と x に apply を適用すると，ゴールが P x になり，trivial で証明が終わる．

```
Lemma all_not_not_ex:(forall x:A, ~P x)
      -> ~exists x:A,P x.
Proof.
intro. intro. destruct H0.
apply (H x). trivial.
Qed.
```

ここでも最後のところで使用した apply (H x). の代わりに，specialize (H x) と仮定を書き換えても矛盾する論理式が仮定に現れるので contradiction で証明できる．

```
Lemma all_not_not_ex:(forall x:A, ~P x)
      -> ~exists x:A,P x.
Proof.
intro. intro. destruct H0.
specialize (H x). contradiction.
Qed.
```

2.2.5　タクティックのカスタマイズ

既存のタクティックを自分用に改変したり、新たに作成することができる．ここでは長い名前のタクティックを自分用の短い名前で定義し直す方法を述べる．タクティックは Ltac という Vernacular コマンドを用いて定義する．例えば、

```
Ltac ok:=trivial;contradiction.
```

とすると、現在のゴールに対して、ok とすると、trivial や contradiction を適用する．なお、セミコロンはタクティックの逐次適用になる．従って、ok を適用すると trivial を試した後で contradiction を行う．

演習問題 3. 次の論理式が証明可能であることを Coq で示せ．

1. $\neg(\exists x P) \to \forall x(\neg P)$
2. $\exists x(\neg P) \to \neg(\forall x P)$
3. $\forall x(P \wedge Q) \leftrightarrow (\forall x P) \wedge (\forall x Q)$（ヒント：仮定に入った forall x, P x/\Q x を x:A に対して destruct するには, destruct (H x) のように書く必要がある．destruct ではなく，specialize で仮定を書き換えることでもできる．
4. $\exists x(P \vee Q) \leftrightarrow (\exists x P) \vee (\exists x Q)$
5. $\exists x \forall y R(x, y) \to \forall y \exists x R(x, y)$

2.2.6 古典論理でないと証明できない例

論理式 $\neg(\forall x P) \to \exists x(\neg P)$ は、証明に古典論理のライブラリー Classical が必要である．

```
Require Import Classical.

Lemma not_all_to_ex_not:~(forall x,P x)
                        ->(exists x,~P x).
Proof.
intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : ~ (forall x : A, P x)
______________________________________(1/1)
exists x : A, ~ P x
```

証明を始める前に，上記の Lemma の記述で補足をする．これまで変数には型を付けてきたが，exists x の x には型が付いていない．これは Coq が x が P の引数になっていることから，型を推論して決定しているからである．実際，右のペインを見てみると変数には型がついているのがわかる．

intro を適用した時点で、ゴールに exists で適用できる項（変数）はなく、また仮定にある否定もはずすことができない．そこで Classical にある二重否定の除去 NNPP を用いて、ゴールに二重否定をつけ、intro で仮定に入れることにする．

```
Require Import Classical.

Lemma not_all_to_ex_not:~(forall x,P x)
                        ->(exists x,~P x).
Proof.
intro. apply NNPP. intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : ~ (forall x : A, P x)
H0 : ~ (exists x : A, ~ P x)
______________________________________(1/1)
False
```

これで仮定 H を適用することが可能になる．

```
Require Import Classical.

Lemma not_all_to_ex_not:~(forall x,P x)
                        ->(exists x,~P x).
Proof.
intro. apply NNPP. intro. apply H.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : ~ (forall x : A, P x)
H0 : ~ (exists x : A, ~ P x)
______________________________________(1/1)
forall x : A, P x
```

ここで intro を行うと、また適用できるタクティックがなくなるので再度 NNPP を適用して intro を行う．

```
Require Import Classical.

Lemma not_all_to_ex_not:~(forall x,P x)
                        ->(exists x,~P x).
Proof.
intro. apply NNPP. intro. apply H.
intro;apply NNPP;intro.
```

```
1 subgoal
A : Type
P, Q : A -> Prop
R : A -> A -> Prop
t : A
H : ~ (forall x : A, P x)
H0 : ~ (exists x : A, ~ P x)
x : A
H1 : ~ P x
______________________________________(1/1)
False
```

最後に仮定 H0 をゴールに適用すると、最初に適用できなかった exists が使える形になっていることがわかる．

```
Lemma not_all_to_ex_not:~(forall x,P x)
                        ->(exists x,~P x).
Proof.
intro. apply NNPP. intro. apply H.
intro;apply NNPP;intro.
apply H0;exists x;ok.
Qed.
```

二重否定の除去 NNPP の後に，intro を行うことは，背理法で証明することに相当する．これは，LK で許容される推論規則

$$\frac{\neg A, \Gamma \vdash \bot}{\Gamma \vdash A}$$

に対応する．次のようにタクティック hairihou を定義しておくと，通常の証明と同じ感覚で背理法を利用することができる．

```
Ltac hairihou:=apply NNPP;intro.
```

演習問題 4. 次の論理式が証明可能であることを Coq で示せ．

1. $\neg\neg\exists x P \to \exists x(\neg\neg P)$
2. $\neg\exists x(\neg P) \to \forall x P$
3. $\exists x A \leftrightarrow \neg\forall x \neg A$

4. $\exists x(P(x) \leftrightarrow \forall y P(y))$

演習問題 5. セクションを閉じて，ファイルを保存せよ．

最後にこれまで紹介してこなかった強力なタクティックの intuition について説明する．intuition はライブラリの Tauto で定義されていて，Coq を起動すると自動的に読み込まれる．intuition はゴールや仮定からの推論が「明らか」なときに使えるタクティックで，これまでいろいろなタクティックを説明するためにあえて使ってこなかったが，命題論理のところで扱った例の多く（特に LJ で証明できるもの）は，intuition. だけで証明できたり，証明が短縮できる．intuition がどのように定義されているかを説明しないが，自分で新たにタクティックを定義しようとする場合，よい参考になる．

3 集合の基本的性質の証明

3.1 ライブラリ Ensembles を利用した集合の性質の証明

3.1.1 ライブラリ Ensembles

Coq を用いて数学的な主張を証明するための例としてここでは集合のかんたんな性質を証明して行くことにする．集合を Coq でどのように扱うかであるが，（Coq には Set という型があるが）Library の Ensembles を使うことにする．Ensembles.v は Coq のインストールフォルダ内にある．

Ensembles では次のように集合（Ensemble）を定義している．

```
Variable U : Type.
Definition Ensemble := U -> Prop.
```

U が全体集合で，その部分集合を Ensemble と定義していると考えることができる．Ensemble は，この定義だと前節の述語論理での述語の扱いと変わらないが，これは U 上の述語とその述語をみたす元の集合の関係

$$R(a) \leftrightarrow a \in \{x \in U \mid R(x)\} \qquad (*)$$

を考えれば，同じ扱いでよいことがわかる．次は集合の要素関係 $\in$ の定義である．

```
Definition In (A:Ensemble) (x:U) : Prop := A x.
```

これも上記の (*) を考えればこの定義でよいことがわかる．次は集合の包含関係の定義である．

```
Definition Included (B C:Ensemble) :  Prop :=
                      forall x:U, In B x -> In C x.
```

これは通常の包含関係の定義である．次は全体集合と空集合の定義である．

```
Inductive Full_set :  Ensemble :=
              Full_intro :  forall x:U, In Full_set x.
Inductive Empty_set :  Ensemble :=.
```

ここで Inductive は，対象を帰納的に定義する場合の用語である．Full_set の定義にある Full_intro はこの集合のコンストラクタ（この集合の元は何であるかを示すもの）で，Full_set に関わる事実を証明する際に，この名前を用いて apply Full_intro のように使うことができる．一方，Empty_set には何も書かれていない．これはコンストラクタがない（つまり何もないから空である）ことを表している．

空集合と全体集合を Inductive ではなく Definition で定義すると次のようにできる．この二つの定義の同等性は後で証明する．

```
Definition Full_set :  Ensemble := fun x:U=>True.
Definition Empty_set :  Ensemble := fun x:U=>False.
```

fun x:A=>y は写像の対応関係でよく使われる $x \mapsto y$ を表している．従って，Definition による定義は，Ensemble の型である U->Prop が，U から Prop への写像であると考えると自然な定義である．Inductive で定義した場合と Definition で定義した場合で証明で使うタクティックが変わるが，説明は具体的な証明の中で行う．

集合に関する演算の和集合 (union)∪，共通部分 (intersection)∩ は次のように Inductive に定義する．

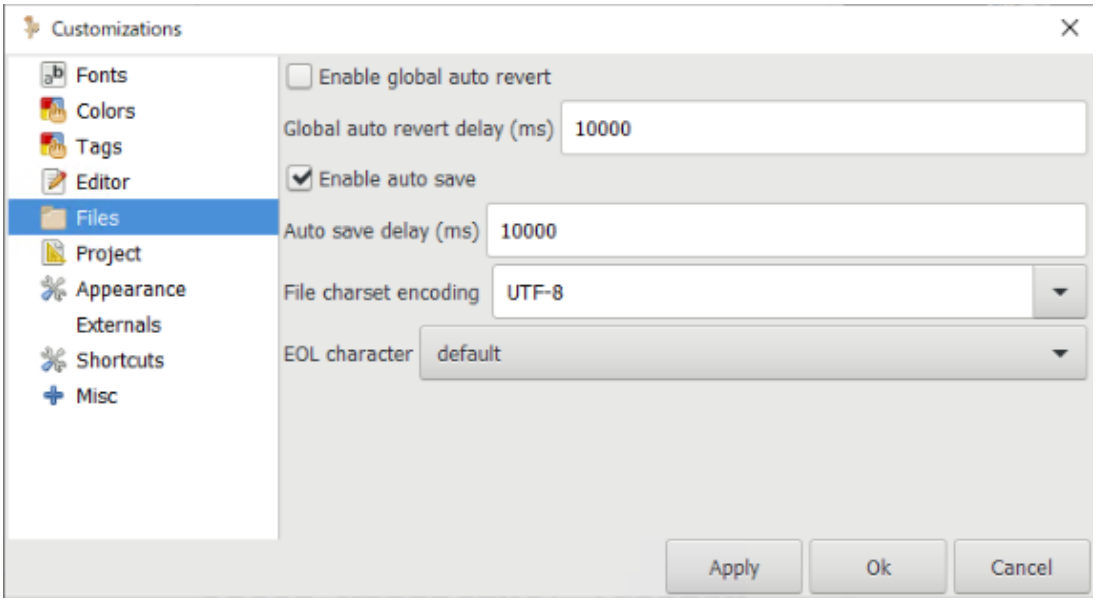
```
Inductive Union (B C:Ensemble) :  Ensemble :=
    | Union_introl :  forall x:U, In B x -> In (Union B C) x
    | Union_intror :  forall x:U, In C x -> In (Union B C) x.
Inductive Intersection (B C:Ensemble) :  Ensemble :=
  Intersection_intro :
      forall x:U, In B x -> In C x -> In (Intersection B C) x.
```

集合 B と C の和集合の元は，B の元であるか C の元のいずれかであることと定義され，共通部分は B の元かつ C の元であることと定義されている．ライブラリ Ensembles で定義されている他の演算の定義については，集合の基本的な性質を証明しながら説明する．

3.1.2 基本的性質の証明

ここで証明する集合の性質は，集合と写像に関する授業資料[*12]から抜粋したもので，付録に掲載をしている．定理や問題番号はそれに沿っているので適宜参照してほしい．なお，この先 Unicode の記号を利用するため，メニューの Edit->Preferences を選び，Files の File charset encoding が UTF-8 になっているか確認し，UTF-8 でない場合は UTF-8 を選択する必要がある．（選択後 OK をクリックする．）

[*12] http://herb.h.kobe-u.ac.jp/nst/



最初に Ensembles をインポートする．背理法を使うことがあるので，Classical もインポートする．また，論理のところで定義した 2 つのタクティックも使えるようにする．また，集合の型を Ensemble と書くのは面倒なので，略記を用意する．

```
Require Import Ensembles.
Require Import Classical.

Section SetTheory.

Ltac ok:=trivial;contradiction.
Ltac hairihou:=apply NNPP;intro.

Variable U:Type.

Notation Shugo:=(Ensemble U).
```

Coq で証明するときに見やすいように，最初に Notation で略記を定義する．2 項演算を定義するときは，結合の強さと結合の順序を定義する必要がある．ここでは複合命題を表現するときに，仮定やゴールの表示でカッコを省略せずそのまま表示するようにするためにすべて no associativity で指定しておく．また，この後，証明する命題の主張を簡潔にするため，使用する変数も定義する．

```
Notation "x ∈ A":=(In U A x)(at level 55,no associativity).
Notation "A ⊆ B":=(Included U A B)(at level 55, no associativity).
Notation "A ∩ B":=(Intersection U A B)(at level 54, no associativity).
Notation "A ∪ B" :=(Union U A B)(at level 54, no associativity).
Notation ø:=(Empty_set U).
Notation Ω:=(Full_set U).

Variable A B C D:Shugo.
```

集合の元であるかないかを示すには排中律が必要になる．

```
Lemma in_or_not:forall A,
            forall x,(x ∈ A)\/~(x ∈ A).
Proof.
intros;apply classic.
Qed.
```

x が集合 A の元であるかどうかで場合分けするには，この in_or_not が必要になる[*13]．Coq の章の述語論理のところで述べたように，Coq が型を推論できる部分は表現を簡潔にするために上記のように型を省略することにする．以下，付録にある集合や写像の性質を Coq を用いて証明をしていく．

命題 A.1 $A \subseteq B$ かつ $B \subseteq C$ ならば $A \subseteq C$.

最初に部分集合の包含関係 Included を定義に従って分解する．これは unfold Included を行う．

```
Lemma bubun_transitive:(A⊆B)/\(B⊆C)->A⊆C.
Proof.
unfold Included.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
______________________________________(1/1)
(forall x : U, (x ∈ A) -> x ∈ B) /\
(forall x : U, (x ∈ B) -> x ∈ C) ->
forall x : U, (x ∈ A) -> x ∈ C
```

次に intros. を行う．これで現在の状況は

```
Lemma bubun_transitive:(A⊆B)/\(B⊆C)->A⊆C.
Proof.
unfold Included. intros.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
H : (forall x : U, (x ∈ A) -> x ∈ B) /\
    (forall x : U, (x ∈ B) -> x ∈ C)
x : U
H0 : x ∈ A
______________________________________(1/1)
x ∈ C
```

である．あとは，destruct H を行い分解された仮定を使えば証明が終了する．

```
Lemma bubun_transitive:(A⊆B)/\(B⊆C)->A⊆C.
Proof.
unfold Included. intros.
destruct H. apply H1. apply H. ok.
Qed.
```

Included の分解はよく行うので，タクティックとして定義する．

```
Ltac bubun:=unfold Included;intros.
```

定理 A.2 空集合は任意の集合の部分集合である．すなわち，任意の集合 A に対し，$\varnothing \subseteq A$ が成り立つ．

証明は，intro. bubun. destruct H. で終わる．

[*13] この資料で使用するのは．問題 A.8 の証明のところである．

```
Lemma empty_bubun:forall A,ø⊆A.
Proof.
intro. bubun. destruct H.
Qed.
```

ここで扱う集合は全体集合の部分集合であるのでそれも示示す.

```
Lemma bubun_full:forall A,A⊆Ω.
Proof.
intro;bubun. apply Full_intro.
Qed.
```

証明で注意すべき点は，intro;bubun. を行った後，Full_set の定義のコンストラクタである Full_intro を使うことである．apply Full_intro とすると，結論が一気に証明できる．なお，split は intros した後に，ただ一つのコンストラクタを適用することなので，この Lemma は split とするとすぐに証明が終わる．

```
Lemma bubun_full:forall A,A⊆Ω.
Proof.
split.
Qed.
```

今後集合の同等性を示すことが多くなるが，それには外延性の定理を用いる．

```
Axiom Extensionality_Ensembles :
          forall A B:Ensemble, Same_set A B -> A = B.
```

ここで，Same_set の定義は，

```
Definition Same_set (B C:Ensemble) :  Prop :=
          Included B C /\ Included C B.
```

であるから，通常の外延性定理 $\forall A, B[A \subseteq B \wedge B \subseteq A \rightarrow A = B]$ である．ゴールに入る/\を分解することまで含め，次のようにタクティック seteq を定義する．今後集合の同等性を示す場合は最初にこの seteq を使用する．

```
Ltac seteq:=apply Extensionality_Ensembles;unfold Same_set;split.
```

定理 A.3 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A \cup A = A$
2. $A \cup B = B \cup A$
3. $A \cup (B \cup C) = (A \cup B) \cup C$
4. $A \subseteq A \cup B, B \subseteq A \cup B$
5. $A, B \subseteq C \rightarrow A \cup B \subseteq C$

証明で注意することは和集合のコンストラクタ Union_introl と Union_intror の使い

方だけであるが，left や right は intros した後にそれぞれコンストラクタの 1 と 2 を適用するものなので，left と right を使用するほうが簡単である．より一般的には，n 番目のコンストラクタを適用する constructor n を用いて，constructor 1 や constructor 2 でも構わない．証明を後で見直す時に，Union_introl と Union_intror と書いてある方が見やすいということだけである．

seteq の後最初のゴールは簡単で 2 つ目のゴールの証明で bubun を適用すると次のようになる．

```
Lemma union_id:A∪A=A.
Proof.
seteq. bubun. destruct H. ok. ok.
bubun.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
x : U
H : x ∈ A
_____________________________________(1/1)
x ∈ A ∪ A
```

ここで apply Union_introl あるいは left を適用する[*14]．するとゴールが x ∈ A に変換されて証明ができる．

```
Lemma union_id:A∪A=A.
Proof.
seteq. bubun. destruct H. ok. ok.
bubun. apply Union_introl. ok.
Qed.
```

A.3 の 3 の証明では，仮定に∪がある場合は destruct で分解される（\/と同じ）ことと Union_introl と Union_intror の使い分けに気をつける．

```
Lemma union_comm:A∪B=B∪A.
Proof.
seteq. bubun. destruct H.
apply Union_intror. ok. apply Union_introl. ok.
bubun. destruct H.
apply Union_intror. ok. apply Union_introl. ok.
Qed.
```

演習問題 6. 定理 A.3 の 4 と 5 及び問題 A.4 の左から右を証明せよ．

問題 A.4 の右から左を示すときに，タクティックの新しい使い方が必要になる．intro を行うと次の状況になる．

```
Lemma probA4b:(A∪B)=B->(A⊆B).
Proof.
intro.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
H : A ∪ B = B
_____________________________________(1/1)
A ⊆ B
```

ここで仮定に対して，destruct H を適用すると，ゴールが A ⊆ A ∪ B に変わる．これは，仮定 H を利用してゴールの B を A ∪ B に書き換えたことになる．明示的に等式を利用し

[*14] この場合は apply Union_intror あるいは right でも良い．

て書き換えるタクティックは rewrite である．rewrite は仮定にある等式の左式を右式に置き換えるが，今回のように右式を左式に置き換えるには rewrite <-を用いる．rewrite H ではなく，rewrite <- H とするとゴールが確かに変わる．

```
Lemma probA4b:(A∪B)=B->(A⊆B).
Proof.
intro. rewrite <- H.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
H : A ∪ B = B
_____________________________________(1/1)
A ⊆ A ∪ B
```

あとは定理 A.3 の 4 の最初の式を用いれば良い．（以下の証明では，定理 A.3 の 4 の最初の式の命題の名前を bubun_unionl としている．

```
Lemma probA4b:(A∪B)=B->(A⊆B).
Proof.
intro. rewrite <- H.
apply bubun_unionl.
Qed.
```

定理 A.5 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A \cap A = A$
2. $A \cap B = B \cap A$
3. $A \cap (B \cap C) = (A \cap B) \cap C$
4. $A \cap B \subseteq A, A \cap B \subseteq B$
5. $C \subseteq A, B \to C \subseteq A \cap B$
6. $A \subseteq B \leftrightarrow A \cap B = A$

証明は，使うタクティックが Intersection_intro であることを除けば，和集合と同じようにできる．この場合も split がコンストラクタの 1 を適用するものなので，apply Intersection_intro よりも split を使用する方が簡単である．

演習問題 7. 定理 A.5 を証明せよ．

Ensembles では，Disjoint は次のように定義されている．

```
Inductive Disjoint (B C:Ensemble) :  Prop :=Disjoint_intro :
      forall x:U, ~In (Intersection B C) x) -> Disjoint B C.
```

すなわち，B ∩ C に元がないときに，Disjoint B C と定義している．よって，Disjoint ならば共通部分は空集合になるが，それを示すのが次の Lemma である．

```
Lemma Kuu:Disjoint U A B<->(A∩B)=∅.
Proof.
split. intros. seteq. bubun. destruct H.
specialize(H x). ok. apply empty_bubun.
intro. apply Disjoint_intro.
intro. intro. rewrite H in H0.
destruct H0.
Qed.
```

specialize で仮定を書き換えるのは論理のところで行ったのと同じである．ここでも apply Disjoint_intro ではなく split を使うことができる．rewrite H in H0 は等式 H を用いて，仮定 H1 を書き換えるときの書き方である．

演習問題 8. 定理 A.6 と問題 A.7 の左から右を証明せよ．

次に問題 A.7 の右から左を証明する．

```
Lemma probA7b:(A∪B)=(B∩C)->(A⊆B)/\(B⊆C).
Proof.
intro. split. bubun.
```

```
2 subgoals
U : Type
A, B, C, D : Shugo
H : A ∪ B = B ∩ C
x : U
H0 : x ∈ A
_____________________________________(1/2)
x ∈ B
_____________________________________(2/2)
B ⊆ C
```

この時点で仮定とゴールを見ると,，仮定 H0 より x ∈ A ∪ B が成り立ち，さらに仮定 H より x ∈ B ∩ C となり x0 ∈ B が証明できることがわかる．しかし，これを実現するためには，x ∈ A ∪ B が仮定にないといけない．このような証明の途中に必要な事実を入れるには assert あるいは cut を用いる．

```
assert (x ∈ (A ∪ B)).
```

を評価すると，ゴールが x ∈ A ∪ B になる．

```
Lemma probA7b:(A∪B)=(B∩C)->(A⊆B)/\(B⊆C).
Proof.
intro. split. bubun.
assert (x ∈ A∪B).
```

```
3 subgoals
U : Type
A, B, C, D : Shugo
H : A ∪ B = B ∩ C
x : U
H0 : x ∈ A
_____________________________________(1/3)
x ∈ A ∪ B
_____________________________________(2/3)
x ∈ B
_____________________________________(3/3)
B ⊆ C
```

証明が終了するとこれが仮定に入る．

```
Lemma probA7b:(A∪B)=(B∩C)->(A⊆B)/\(B⊆C).
Proof.
intro. split. bubun.
assert (x ∈ A∪B).
apply Union_introl. ok.
```

```
2 subgoals
U : Type
A, B, C, D : Shugo
H : A ∪ B = B ∩ C
x : U
H0 : x ∈ A
H1 : x ∈ A ∪ B
_____________________________________(1/2)
x ∈ B
_____________________________________(2/2)
B ⊆ C
```

後はこれまでと同様に証明を行えばよい．cut の場合は，証明の順序が逆になる．cut (x ∈ (A ∪ B))． とすると，さきに，これを用いて証明するモードになり，終わると x ∈ A ∪ B を証明するモードに入る．

いずれを用いても，後は仮定に入れた x ∈ (A ∪ B) により証明ができる．

```
Lemma probA7b:(A∪B)=(B∩C)->(A⊆B)/\(B⊆C).
Proof.
intro. split. bubun.
assert (x ∈ A∪B).
apply Union_introl. ok.
rewrite H in H1. destruct H1. ok.
bubun. assert (x ∈ A∪B).
apply Union_intror. ok.
rewrite H in H1. destruct H1. ok.
Qed.
```

差集合 Seminus は Definition で次のように定義されていて，和集合や共通部分のようなコンストラクタがないため，unfold を多用する必要がある．

```
Definition Setminus (B C:Ensemble) :  Ensemble :=
          fun x:U => In B x /\~In C x.
```

コンストラクタの代わりになる Lemma を証明しておくと便利である．差集合の記号は以下のように定義する．

```
Notation "A \ B":=(Setminus U A B)(at level 60, no associativity).
```

```
Lemma setminus:forall A B,forall x,
       (x ∈ A)->(~(x ∈ B)->(x ∈ (A\B))).
Proof.
intros. unfold In. unfold Setminus.
split. ok. ok.
Qed.
```

差集合についてもこれまでの同様に証明ができるがこの節の最初に説明したように in_or_not を使う必要のあるものがいくつかある．その一つが問題 A.8 の $(A - B) \cup (A \cap B) = A$ である．

この問題は $A \subseteq (A - B) \cup (A \cap B)$ を示す時に，$x \in B$ の場合と $x \notin B$ の場合に分けて証明する必要が出てくる．

```
Lemma probA8:((A \ B)∪(A∩B))=A.
Proof.
seteq. bubun. destruct H. destruct H. ok.
destruct H. ok.
bubun.
```

```
1 subgoal
U : Type
A, B, C, D : Shugo
x : U
H : x ∈ A
_____________________________________(1/1)
x ∈ (A \ B) ∪ (A ∩ B)
```

このときのゴールは x ∈ (A \ B) ∪ (A ∩ B) であるが，仮定に B に関する情報がないため，和集合のどちらに入るか決定することができない．そこで

```
destruct (in_or_not B x)
```

とすることで x ∈ B の場合と~(x ∈ B) の場合に分けることができて証明が完成する．

```
Lemma probA8:((A \ B)∪(A∩B))=A.
Proof.
seteq. bubun. destruct H. destruct H. ok.
destruct H. ok.
bubun. destruct (in_or_not B x).
apply Union_intror. split. ok. ok.
apply Union_introl. apply setminus.
ok. ok.
Qed.
```

ここで前に言及した Full_set と Empty_set を Inductive と Definition で定義したものの同等性を示しておく．具体的には次の 2 つの Lemma になる．証明は特に難しいことはない．これらにより，Full_set U は，fun x:U => True であり，Empty_set は fun x:U => False であることがわかる．

```
Lemma fullset:forall x, (Full_set U x)<->True.
Proof.
intro. split. intro. ok. intro. apply Full_intro.
Qed.

Lemma emptyset:forall x, (Empty_set U x)<->False.
Proof.
intro. split. intro. destruct H. intro. ok.
Qed.
```

演習問題 9. 付録 A の残りの定理と問題を証明しなさい．

最後に Section を終了する．

```
End SetTheory.
```

集合に関してここまで証明したものを Shugo.v として保存する．

3.2 ライブラリーの作成と読み込み

3.2.1 ライブラリーの作成

自分で定理を証明したものをライブラリーとして作成するには，ファイル（.v ファイル）を開いた状態でメニューの Compile から Compile buffer を選ぶ．これで同じディレクトリに.vo ファイル（と.glob ファイル）が生成される．ライブラリを読み込むときに使用するの

はこの.vo ファイルである．

前節で作成した Shugo.v を Compile して，新しいライブラリを作成する．作成したライブラリを読み込んだときに，新たに作成したタクティックと集合演算の記法がそのまま使えるように，Shugo.v の End SetTheory の後に以下のものを記述した後に Compile を行い，ライブラリを作成する．

```
Ltac ok:=trivial;contradiction.
Ltac hairihou:=apply NNPP;intro.
Ltac seteq:=apply Extensionality_Ensembles;unfold Same_set;split.
Ltac bubun:=unfold Included;intros.
Notation "x ∈ A":=(In _ A x)(at level 50,no associativity).
Notation "A ⊆ B":=(Included _ A B)(at level 100,no associativity).
Notation "A ∪ B" :=(Union _ A B)(at level 80, no associativity).
Notation "A ∩ B":=(Intersection _ A B)(at level 80, no associativity).
Notation "A \ B":=(Setminus _A B)(at level 60, no associativity).
Notation ø :=(Empty_set _).
Notation Ω:=(Full_set _).
```

3.2.2 ライブラリの読み込み

作成したライブラリを読み込むには，ライブラリーの置かれているフォルダーを LoadPath で指定してから，Require Import を用いる．例えば，hogehoge.vo を置いたフォルダの絶対パスを PATH とするとき，hogehoge を読み込むには

```
Add LoadPath "PATH".
Require Import hogehoge.
```

とする．

3.3 集合族

自分で作成したライブラリーの利用の演習も兼ねて，集合族のかんたんな性質を証明する．まず最初に必要なライブラリー Shugo を読み込む．Windows 10 の場合は

```
Add LoadPath "C:\Users\makoto\Documents\coq".

Require Import Ensembles.
Require Import Classical.
Require Import Shugo.
```

のように書く．ここでは C:\Users\makoto\Documents\coq にライブラリーがおかれていることにしている．Mac OS X の場合は

```
Add LoadPath "/Users/makoto/Desktop/Coq".

Require Import Ensembles.
Require Import Classical.
Require Import Shugo.
```

のように書く．ここでは/Users/makoto/Desktop/Coq にライブラリーがおかれていることにしている．

一般に集合 Λ の元 λ に対して，集合 F_λ が決められているとき，$\{F_\lambda \mid \lambda \in \Lambda\}$ を Λ を添字の集合とする集合族とよぶ．添字の集合は空ではないと仮定する．

ここでは，集合族の Coq における定義を次のように考えることにする．

```
Variable U:Type.
Notation Shugo:=(Ensemble U).

Variable K:Type.
Definition Fam:= K->Shugo.
```

集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ に対し，F_λ のいずれかの元となるようなものの全体，すなわち $\exists\lambda \in \Lambda[a \in F_\lambda]$ をみたす a の全体を $\{F_\lambda \mid \lambda \in \Lambda\}$ の和集合とよび，$\bigcup\{F_\lambda \mid \lambda \in \Lambda\}$ あるいは $\bigcup_{\lambda\in\Lambda} F_\lambda$ で表す．また，集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ に対し，F_λ のすべてに共通な元の全体，すなわち $\forall\lambda \in \Lambda[a \in F_\lambda]$ をみたす a の全体を $\{F_\lambda \mid \lambda \in \Lambda\}$ の共通部分とよび，$\bigcap\{F_\lambda \mid \lambda \in \Lambda\}$ あるいは $\bigcap_{\lambda\in\Lambda} F_\lambda$ で表す．

このことを踏まえ，集合族の和集合と共通部分の Coq における定義を次のようにする．

```
Inductive UnionF (X:Fam):Shugo:=
unionf_intro:forall x:U,(exists n:K,(x ∈ (X n)))-> x ∈ (UnionF X).

Inductive InterF (X:Fam) :Shugo :=
interf_intro:forall x:U,(forall n:K,(x ∈ (X n)))-> x ∈ (InterF X).
```

集合族については次の定理 B.1, B.2 が成り立ちますが，これを Coq で確かめてみる．

定理 B.1 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ と集合族の構成元 F_{λ_0} に対し，次が成立する．

1. $F_{\lambda_0} \subseteq \bigcup_{\lambda\in\Lambda} F_\lambda \quad (\lambda_0 \in \Lambda)$
2. $\bigcap_{\lambda\in\Lambda} F_\lambda \subseteq F_{\lambda_0} \quad (\lambda_0 \in \Lambda)$

定理 B.2 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$,$\{G_\lambda \mid \lambda \in \Lambda\}$ に対し，次が成立する．

1. $\bigcap_{\lambda\in\Lambda} F_\lambda \subseteq \bigcap_{\lambda\in\Lambda} G_\lambda \leftrightarrow \forall\lambda \in \Lambda[F_\lambda \subseteq G_\lambda]$

2. $\bigcup_{\lambda\in\Lambda} F_\lambda \subseteq \bigcup_{\lambda\in\Lambda} G_\lambda \leftrightarrow \forall\lambda\in\Lambda[F_\lambda \subseteq G_\lambda]$

この証明は，和集合と共通部分のコンストラクタ `unionf_intro` と `interf_intro` を使うところに注意をすれば特に難しいことはない．

```
Lemma mem_unionf:forall F:Fam,forall n:K,
                          ((F n) ⊆ UnionF F).
Proof.
intros. bubun. apply unionf_intro.
exists n. ok.
Qed.

Lemma mem_interf :forall F:Fam, forall n:K,
                          ((InterF F) ⊆ (F n)).
Proof.
intros. bubun. destruct H. apply H.
Qed.
```

```
Lemma unionf_inc:forall F G:Fam,(forall n:K,
          (F n)⊆(G n))->(UnionF F)⊆(UnionF G).
Proof.
intros. bubun. destruct H0. destruct H0.
apply unionf_intro. exists x0. apply H. ok.
Qed.

Lemma interf_inc:forall F G:Fam,(forall n:K,
          (F n)⊆(G n))->(InterF F)⊆(InterF G).
Proof.
intros. bubun. apply interf_intro. intro.
destruct H0. specialize (H n).
apply H. apply H0.
Qed.
```

次に，定理 B.3 を確かめてみる．

定理 B.3 集合族 $\{F_\lambda \mid \lambda\in\Lambda\}$ と集合 A に対し，次が成立する．

$$A\cup(\bigcap_{\lambda\in\Lambda} F_\lambda) = \bigcap_{\lambda\in\Lambda}(A\cup F_\lambda)$$

この定理を Coq で表現するためには，集合族 $\{A\cup F_\lambda\}_{\lambda\in\Lambda}$ を定義する必要がある．この定義は，`Definition` で行う．各 $\lambda\in\Lambda$ に対して，$A\cup F_\lambda$ を対応させることに注意する．

```
Definition UFam (F:Fam)(X:Shugo):Fam:=fun n:K=> X∪(F n).
```

証明中で `UFam` を定義に従って書き換えるとき，ゴールにある場合は `unfold UFam.` とし，仮定 `H` にある場合は `unfold UFam in H.` とする．

定理 B.3 を証明する際に，問題となる箇所は次の部分である．

```
Lemma union_interf:forall (F:Fam)(X:Shugo),
          (X∪(InterF F))=(InterF (UFam F X)).
Proof.
intros. seteq. bubun. split. intro.
destruct H. unfold UFam. left. ok.
destruct H. unfold UFam. right. apply H.
bubun. destruct H. unfold UFam in H.
```

```
1 subgoal
U : Type
K : Type
F : Fam
X : Shugo
x : U
H : forall n : K, x ∈ (X ∪ F n)
______________________________________(1/1)
x ∈ (X ∪ InterF F)
```

ゴールは和集合なので，どちらに入るか決めないといけないが，この時点の仮定ではどちらに入るか決めることはできない．そこで in_or_not を用いて，$x \in X$ の場合と $x \notin X$ の場合分けを行う．後は特に難しいところはない．

```
Lemma union_interf:forall (F:Fam)(X:Shugo),
          (X∪(InterF F))=(InterF (UFam F X)).
Proof.
intros. seteq. bubun. split. intro.
destruct H. unfold UFam. left. ok.
destruct H. unfold UFam. right. apply H.
bubun. destruct H. unfold UFam in H.
destruct (in_or_not U X x).
left. ok.
right. split. intro. specialize(H n).
destruct H. ok. ok.
Qed.
```

演習問題 10. 問題 B.4 を示せ．ただし，集合族 $\{A \cap F_\lambda\}_{\lambda\in\Lambda}$ は次のように定義する．証明中で IFam を書き換えるは，UFam と同様に unfold を使用する．

```
Definition IFam (F:Fam)(X:Shugo):Fam:=fun n:K=> (F n)∩X.
```

また，添字集合が空でないことを明示するために，以下の宣言をする．

```
Variable n:K.
```

最後にファイルをファイル名を Family.v にしてコンパイルせよ．コンパイルしたファイルは，同値関係のところで Import する．

4　写像の性質の証明

ここで証明する写像の性質は，集合と同様 http://herb.h.kobe-u.ac.jp/nst/から抜粋したもので，付録 C に掲載をしているものである．

4.1　像と逆像の定義

集合族の性質の証明と同様に，ライブラリをインポートする．

```
Add LoadPath "/Users/makoto/Desktop/Coq".

Require Import Ensembles.
Require Import Classical.
Require Import Shugo.
```

写像は定義域と値域の集合を型としては区別する必要があるため，次のように UShugo と VShugo を定義する．Set Implicit Arguments を宣言すると，型が推論できるような引数

を省略することができる．

```
Section Image.

Set Implicit Arguments.

Variables U V: Type.

Notation UShugo:=(Ensemble U).
Notation VShugo:=(Ensemble V).
```

$f : A \to B$ と $C \subseteq A, D \subseteq B$ に対し，f による C の像 $f(C)$ と f による D の逆像 $f^{-1}(D)$ は次のように定義される．

$$f(C) = \{f(a) \mid a \in C\},\ f^{-1}(D) = \{a \in A \mid f(a) \in D\}$$

よって，Coq では次のように定義するのが自然である．

```
Inductive Im (X:UShugo) (f:U -> V) : VShugo :=
    Im_intro : forall x:U, x ∈ X  ->  (f x) ∈ (Im X f).


Inductive InvIm (Y:VShugo) (f:U -> V) : UShugo :=
    InvIm_intro : forall x:U,
     (f x) ∈ Y -> x ∈ (InvIm Y f) .
```

4.2　像の性質の証明

まず最初に $x \in X$ ならば $f(x) \in f(X)$ であることを示す．証明は直線的にできる．

```
Lemma Im_element:forall (X:UShugo)(f:U->V)(y:V),
(y ∈ (Im X f)) <->exists x:U, (x ∈ X)  /\ y=f x.
Proof.
split. intro. destruct H. exists x.
split. ok. ok.
intro. destruct H. destruct H.
rewrite H0. split. ok.
Qed.
```

定理 C.3 $f : U \longrightarrow V$ とする．$A, B \subseteq U$ とするとき，次の関係が成り立つ．

1. $A \subseteq B \to f(A) \subseteq f(B)$
2. $f(A \cup B) = f(A) \cup f(B)$
3. $f(A \cap B) \subseteq f(A) \cap f(B)$

順に証明すると次のようになる．いずれも集合で証明したときに用いたことと像のコンス

トラクタから証明ができる．証明中では像のコンストラクタを直接適用しているのではなく split で代用している．特に共通部分に関わるところでは，像のコンストラクタを使うところと共通部分のコンストラクタを使うところをどちらも split で代用しているので，どちらを使用しているのか注意せよ．

```
Lemma Im_subset:forall (A B:UShugo)(f:U->V),
  (A ⊆ B)-> (Im A f) ⊆ (Im B f).
Proof.
intros. bubun. destruct H0. split.
apply H. ok.
Qed.
```

```
Lemma Im_union:forall (A B:UShugo) (f:U->V),
(Im (A ∪ B) f)=((Im A f) ∪ (Im B f)).
Proof.
intros. seteq. bubun. destruct H. destruct H.
left. split. ok. right. split. ok.
bubun. destruct H.
destruct H. split. left. ok.
destruct H. split. right. ok.
Qed.
```

```
Lemma Im_intersection:forall (A B:UShugo)(f:U->V),
  (Im (A ∩ B) f) ⊆ ( (Im A f) ∩ (Im B f)).
Proof.
intros. bubun. destruct H. destruct H.
split. split. ok. split. ok.
Qed.
```

4.3　逆像の性質の証明

次に逆像に関することを証明してみる．まず最初に，$x \in f^{-1}(Y) \leftrightarrow f(x) \in Y$ を示す．証明は逆像のコンストラクタを用いることで容易にできる．ここでも逆像のコンストラクタを指定する代わりに split で代用している．

```
Lemma InvIm_def:forall (Y:VShugo) (x:U) (f:U -> V),
(x ∈ (InvIm Y f) )<-> ( (f x) ∈ Y).
Proof.
split. intro. destruct H. ok.
intro. split. ok.
Qed.
```

次に，$f^{-1}(Y) = f^{-1}(f(U) \cap Y)$ を示す．

```
Lemma InvImSet:forall (Y:VShugo) (f:U -> V),
InvIm Y f=InvIm (Y ∩ (Im (Full_set U) f)) f.
Proof.
intros. seteq. bubun. destruct H. split.  split. ok.
split. split.
bubun. destruct H. split. destruct H. ok.
Qed.
```

定理 C.6 $f : U \longrightarrow V$ とする．$C, D \subseteq U$ とするとき，次の関係が成り立つ．

1. $C \subseteq D \rightarrow f^{-1}(C) \subseteq f^{-1}(D)$

2. $f^{-1}(C \cup D) = f^{-1}(C) \cup f^{-1}(D)$
3. $f^{-1}(C \cap D) = f^{-1}(C) \cap f^{-1}(D)$

1. の証明では，特に難しいところはない．

```
Lemma teiriB61:forall (f:U -> V)(C D:VShugo),
 (C ⊆ D)->((InvIm C f) ⊆ (InvIm D f)).
Proof.
intros. bubun. destruct H0. split. apply H. ok.
Qed.
```

2. は集合の同等性を証明するので，これまで同様にすすめると

```
Lemma teiriB62:forall (f:U -> V)(C D:VShugo),
InvIm (C ∪ D) f=((InvIm C f) ∪ (InvIm D f)).
Proof.
intros. seteq. bubun. destruct H.
```

```
2 subgoals
U, V : Type
f : U -> V
C, D : VShugo
x : U
H : f x ∈ (C ∪ D)
_____________________________________(1/2)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(2/2)
InvIm C f ∪ InvIm D f ⊆ InvIm (C ∪ D) f
```

となる．ここで，$f(x) \in C \cup D \leftrightarrow f(x) \in C \vee f(x) \in D$ を使うために仮定 H を使いたいが，そのまま `destruct H` とすると，$f(x)$ の情報も失われてしまう．

```
Lemma teiriB62:forall (f:U -> V)(C D:VShugo),
InvIm (C ∪ D) f=((InvIm C f) ∪ (InvIm D f)).
Proof.
intros. seteq. bubun. destruct H. destruct H.
```

```
3 subgoals
U, V : Type
f : U -> V
C, D : VShugo
x : U
x0 : V
H : x0 ∈ C
_____________________________________(1/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(2/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(3/3)
InvIm C f ∪ InvIm D f ⊆ InvIm (C ∪ D) f
```

そこで `destruct` ではなく `inversion H.` とする．そうすると，$f(x)$ の情報も保存されるので証明が進む．

```
Lemma teiriB62:forall (f:U -> V)(C D:VShugo),
InvIm (C ∪ D) f=((InvIm C f) ∪ (InvIm D f)).
Proof.
intros. seteq. bubun. destruct H. inversion H.
```

```
3 subgoals
U, V : Type
f : U -> V
C, D : VShugo
x : U
H : f x ∈ (C ∪ D)
x0 : V
H0 : f x ∈ C
H1 : x0 = f x
_____________________________________(1/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(2/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(3/3)
InvIm C f ∪ InvIm D f ⊆ InvIm (C ∪ D) f
```

`destruct` を使う場合は，一旦 $y = f(x)$ とおいて，変数 y にしてから `destruct` することにする．そのために使うのが `remember` タクティックである．`remember (f x) as y` と

することで，仮定に Heqy:　y = f x が追加され，destruct H. の後に rewrite すると，仮定が実質 f x ∈ C と f x ∈ D に場合分けされる．

```
Lemma teiriB62:forall (f:U -> V)(C D:VShugo),
InvIm (C ∪ D) f=((InvIm C f) ∪ (InvIm D f)).
Proof.
intros. seteq. bubun. remember (f x) as y.
destruct H. rewrite <-Heqy in H. destruct H.
```

```
3 subgoals
U, V : Type
f : U -> V
C, D : VShugo
x : U
x0 : V
H : x0 ∈ C
Heqy : x0 = f x
_____________________________________(1/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(2/3)
x ∈ (InvIm C f ∪ InvIm D f)
_____________________________________(3/3)
InvIm C f ∪ InvIm D f ⊆ InvIm (C ∪ D) f
```

しかし，このようなことをしなくても，inversion で簡単にできる．
後は，同様に証明をすすめることできる．

```
Lemma teiriB62:forall (f:U -> V)(C D:VShugo),
InvIm (C ∪ D) f=((InvIm C f) ∪ (InvIm D f)).
Proof.
intros. seteq. bubun. destruct H. inversion H.
left. split. ok.
right. split. ok.
bubun. destruct H. inversion H.
split. left. ok. inversion H.
split. right. ok.
Qed.
```

3. も同様にできる．

```
Lemma teiriB63:forall (f:U -> V)(C D:VShugo),
InvIm (C ∩ D) f=((InvIm C f) ∩ (InvIm D f)).
Proof.
intros. seteq. bubun. destruct H. inversion H.
split. split. ok. split. ok.
bubun. destruct H. split. split.
destruct H. ok.
destruct H0. ok.
Qed.
```

演習問題 11. 問題 C.7 を示せ．なお，U, V を集合として扱う場合は，(Full_set U)，(Full_set V) とすることに注意せよ．

5　関係

この章では関係を扱う．ここで証明する関係の性質は，集合と同様授業資料[*15]から抜粋したもので，付録 D に掲載をしているものである．集合と集合族に関する概念を必要とするのでライブラリ Shugo，Family を読み込む．

```
Add LoadPath "/Users/makoto/Desktop/Coq".

Require Import Ensembles.
Require Import Classical.
Require Import Shugo.
Require Import Family.

Section Relation.

Set Implicit Arguments.

Variable U:Type.
Notation Shugo:=(Ensemble U).
```

5.1　同値関係

2 項関係 R は，次の条件 **R1**,**R2**,**R3** を満たすとき，それぞれ**反射的** (reflexive)，**対称的** (symmetric)，**推移的** (transitive) であるという．

R1: $\forall a \in A[aRa]$
R2: $\forall a, b \in A[aRb \rightarrow bRa]$
R3: $\forall a, b, c \in A[aRb \wedge bRc \rightarrow aRc]$

Coq でも次のように定義する．

```
Definition Reflexive (R:U->(U->Prop)):Prop :=
                              forall x:U, R x x.
Definition Symmetric (R:U->(U->Prop)):Prop :=
                              forall x y:U,R x y->R y x.
Definition Transitive (R:U->(U->Prop)):Prop :=
                              forall x y z:U,R x y->R y z->R x z.
```

反射的，対称的かつ推移的である関係を**同値関係** (equivalence relation) とよぶ．

```
Definition Equivalent_Relation (R:U->(U->Prop)):Prop :=
                        Reflexive R /\Symmetric R /\Transitive R.
```

[*15] http://herb.h.kobe-u.ac.jp/nst/

5.2　同値類

R を A 上の同値関係とする．任意の $a \in A$ に対し，a と同値な関係にある A の元全体を $[a]$ で表すことにする．すなわち，

$$[a] := \{b \in A \mid aRb\}$$

とおく．$[a]$ の形の集合を**同値類** (equivalence class) とよび，a をその同値類の**代表元**とよぶ．代表元の取り方は，一意的には決まらない．Coq で同値類を定義するためには，最初に同値関係を決めておく必要がある．そのために Hypothesis を用いる．

```
Variable  R:U->(U->Prop).

Hypothesis equiv:Equivalent_Relation R.
```

これにより，これ以降 R は同値関係として仮定される．Coq で仮定にある同値関係を反射的，対称的，推移的のそれぞれの関係にわけるタクティック eqrel を次のように定義する．

```
Ltac eqrel:=destruct equiv as [ref symtran];
            destruct symtran as [sym tran].
```

ここで, destruct equiv as [ref symtran] は, Equivalent_relation R を destruct したときに，最初の仮定 Reflexive R に ref というラベルをつけ，残りの Symmetric R /\Transitive R に symtran というラベルをつけることを表す．さらに destruct symtran as [sym tran] により，対称律と推移律の式に sym と tran のラベルがつけられることになる．

その上で，同値類を次のように定義する．

```
Inductive Equiv_class (x:U) : Shugo:=
  eqclass_intro :forall y:U,R x y-> y ∈ (Equiv_class x).
```

xRx であるから，$x \in [x]$ となるがこれを最初に確認する

```
Lemma equivclass_id :forall x:U,
                       x ∈ Equiv_class x.
Proof.
intro.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x : U
______________________________________(1/1)
x ∈ Equiv_class x
```

eqrel で同値関係の性質を 3 つにわける．

```
Lemma equivclass_id :forall x:U,
                       x ∈ Equiv_class x.
Proof.
intro. eqrel.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x : U
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
_____________________________________(1/1)
x ∈ Equiv_class x
```

同値類のコンストラクタ eqclass_intro でゴールを変換する.

```
Lemma equivclass_id :forall x:U,
                       x ∈ Equiv_class x.
Proof.
intro. eqrel.
apply eqclass_intro.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x : U
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
_____________________________________(1/1)
R x x
```

反射律 ref を用いれば証明終了である.

```
Lemma equivclass_id :forall x:U,
                       x ∈ Equiv_class x.
Proof.
intro. eqrel.
apply eqclass_intro. apply ref.
Qed.
```

次に xRy ならば $[x] \subseteq [y]$ を示す.

```
Lemma equivclass_rel : forall x y:U,
  R x y->(Equiv_class x)⊆(Equiv_class y).
Proof.
intros. bubun. apply eqclass_intro.
destruct H0. eqrel.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x, y : U
H : R x y
y0 : U
H0 : R x y0
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
_____________________________________(1/1)
R y y0
```

結論と仮定の H と H0 を見ると，推移律を適用できそうであるが，推移律を適用するには仮定 H の変数の順序を入れ替える必要があるので，sym を用いる.

```
Lemma equivclass_rel : forall x y:U,
  R x y->(Equiv_class x)⊆(Equiv_class y).
Proof.
intros. bubun. apply eqclass_intro.
destruct H0. eqrel.
apply sym in H.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x, y : U
H : R y x
y0 : U
H0 : R x y0
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
______________________________________(1/1)
R y y0
```

これで推移律が適用できるので証明が終わる．

```
Lemma equivclass_rel : forall x y:U,
  R x y->(Equiv_class x)⊆(Equiv_class y).
Proof.
intros. bubun. apply eqclass_intro.
destruct H0. eqrel.
apply sym in H.
apply (tran y x y0). ok. ok.
Qed.
```

tran に引数を与えているのは，どの順で適用するか指示する必要があるからである．

tran に引数が必要な理由は，Transitive の定義が

```
Definition Transitive (R:U->(U->Prop)):Prop :=
                              forall x y z:U,R x y->R y z->R x z.
```

で，apply tran を行うときの仮定とゴールの状況が $\Gamma, R\ y\ x, R\ x\ y0 \vdash R\ y\ y0$ だったからである．そこで，仮定にある H0:R x y0 を intro の逆の操作を行う revert を使い revert H0 とすると，tran を適用できる形になり apply tran. で証明ができる．

```
Lemma equivclass_rel : forall x y:U,
  R x y->(Equiv_class x)⊆(Equiv_class y).
Proof.
intros. bubun. apply eqclass_intro.
destruct H0. eqrel.
apply sym in H.
revert H0.
```

```
1 subgoal
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x, y : U
H : R y x
y0 : U
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
______________________________________(1/1)
R x y0 -> R y y0
```

```
Lemma equivclass_rel : forall x y:U,
  R x y->(Equiv_class x)⊆(Equiv_class y).
Proof.
intros. bubun. apply eqclass_intro.
destruct H0. eqrel.
apply sym in H.
revert H0. apply tran. ok.
Qed.
```

定理 D.1 A 上の同値関係 R の同値類に関して次が成り立つ．

1. $\forall a, a' \in A(aRa' \to [a] = [a'])$
2. $\forall a, a' \in A(\neg(aRa') \to [a] \cap [a'] = \varnothing)$
3. $\bigcup_{a \in A} [a] = A$

1. は演習問題とする．2. を証明する．

```
Lemma equivclass_empty :forall x y:U,
~R x y-> ((Equiv_class x)⋂(Equiv_class y))=∅
Proof.
intros. seteq. bubun. destruct H0.
destruct H0. destruct H1. eqrel.
apply sym in H1.
```

```
2 subgoals
U : Type
R : U -> U -> Prop
equiv : Equivalent_Relation R
x, y : U
H : ~ R x y
y0 : U
H0 : R x y0
H1 : R y0 y
ref : Reflexive R
sym : Symmetric R
tran : Transitive R
_____________________________________(1/2)
y0 ∈ ø
_____________________________________(2/2)
ø ⊆ Equiv_class x ⋂ Equiv_class y
```

ここで absurd (R x y) として，~R x y と R x y のいずれも導かれることを示す．~R x y は明らかで，R x y は H1 を revert してから，tran を apply して証明ができる．あるいは，specialize で仮定を書き換えて矛盾を出す方針でも構わない．

```
Lemma equivclass_empty :forall x y:U,
~R x y-> ((Equiv_class x)⋂(Equiv_class y))=∅
Proof.
intros. seteq. bubun. destruct H0.
destruct H0. destruct H1. eqrel.
apply sym in H1. absurd (R x y). ok.
revert H1. apply tran. ok.
apply empty_bubun.
Qed.
```

3. の証明には，Family.v で定義した集合族の和集合を用いる．3. の左辺はここでの定義に従えば，集合族 $\{[x] \mid x : U\}$ の和集合であるから，最初にこの集合族を次のように定義する．

```
Definition EqFam:U->Shugo :=fun x:U=>Equiv_class x.
```

これを踏まえて，証明すべき命題と証明は次のようにできる．

```
Lemma equivclass_all : (UnionF U U EqFam)=Ω.
Proof.
seteq. apply bubun_full. bubun.
split. unfold EqFam. exists x. apply equivclass_id.
Qed.
```

演習問題 12. 定理 D.1 の 1. を示せ．

参考文献

[1] Software Foundations, B. C. Pierce, C. Casinghino, M. Greenberg, V. Sjŏberg, B. Yorgey, Vol.1, Vol. 2, Vol.3, https://softwarefoundations.cis.upenn.edu（2020 年 6 月 5 日閲覧）（日本語訳）ソフトウェアの基礎, 梅村晃広、片山功士、水野洋樹、大橋台地、増子萌、今井宜洋訳, http://proofcafe.org/sf/（2020 年 6 月 5 日閲覧）

[2] An experiment in modernizing Coq's manual. http://web.mit.edu/cpitcla/www/coq-rst/index.html（2020 年 6 月 5 日閲覧）

[3] 情報科学における論理, 小野寛晰著, 日本評論社

[4] Tactic Index — Coq 8.12.0 documentation, https://coq.inria.fr/refman/coq-tacindex.html, (2020 年 8 月 11 日閲覧)

付録A　集合

A.1　部分集合と集合の相等

A.1.1　部分集合

命題 **A.1.** $A \subseteq B$ かつ $B \subseteq C$ ならば $A \subseteq C$

A.1.2　集合の相等

$A \subseteq B$ かつ $B \subseteq A$ ならば，$\forall x[x \in A \leftrightarrow x \in B]$ が成り立つから外延性の公理より，$A = B$ である．

A.2　空集合

定理 **A.2.** 空集合は任意の集合の部分集合である．すなわち，任意の集合 A に対し，$\varnothing \subseteq A$ が成り立つ．

A.3　集合の演算

A.3.1　和集合

定理 **A.3.** 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A \cup A = A$
2. $A \cup B = B \cup A$
3. $A \cup (B \cup C) = (A \cup B) \cup C$
4. $A \subseteq A \cup B, B \subseteq A \cup B$
5. $A, B \subseteq C \rightarrow A \cup B \subseteq C$

問題 **A.4.** $A \subseteq B \leftrightarrow A \cup B = B$ を示せ．

A.3.2　共通部分

定理 **A.5.** 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A \cap A = A$
2. $A \cap B = B \cap A$
3. $A \cap (B \cap C) = (A \cap B) \cap C$
4. $A \cap B \subseteq A, A \cap B \subseteq B$
5. $C \subseteq A, B \rightarrow C \subseteq A \cap B$
6. $A \subseteq B \leftrightarrow A \cap B = A$

$A \cap B = \varnothing$ のとき，A と B は **互いに素** (mutually disjoint) であるとよばれる．

A.3.3　和集合と共通部分の関係

定理 A.6. 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
2. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
3. $A \cup (A \cap B) = A \quad , \quad A \cap (A \cup B) = A$

問題 A.7. $A \subseteq B \subseteq C \leftrightarrow A \cup B = B \cap C$ を示せ．

A.3.4　差集合

定義より $A - B \subseteq A, (A - B) \cap B = \varnothing$ が常に成り立つ．

問題 A.8. $(A - B) \cup (A \cap B) = A$ を示せ．

命題 A.9. $A \subseteq C$ かつ $D \subseteq B$ ならば $A - B \subseteq C - D$

問題 A.10. $A \subseteq B \leftrightarrow A - B = \varnothing$ であることを示せ．

問題 A.11. $A - (A - B) = A \cap B$ を示せ．

問題 A.12. $A, B \subseteq C$ とする．$A \cap B = \varnothing \leftrightarrow A \subseteq C - B$ を示せ．

問題 A.13. $(A - C) \subseteq (A - B) \cup (B - C)$ を示せ．

定理 A.14 (ド・モルガンの法則)**.** 集合 A, B, C に対し，次の関係式が成り立つ．

1. $A - (B \cup C) = (A - B) \cap (A - C)$
2. $A - (B \cap C) = (A - B) \cup (A - C)$

問題 A.15. 次を示せ．

1. $(A \cup B) - C = (A - C) \cup (B - C)$
2. $(A \cap B) - C = (A - C) \cap (B - C)$
3. $(A \cap C) - B = (A - B) \cap C$
4. $(A \cup C) - B \subseteq (A - B) \cup C$

付録B　集合族

定理 B.1. 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ と集合族の構成元 F_{λ_0} に対し，次が成立する．

1. $F_{\lambda_0} \subseteq \bigcup_{\lambda\in\Lambda} F_\lambda \quad (\lambda_0 \in \Lambda)$

2. $\bigcap_{\lambda\in\Lambda} F_\lambda \subseteq F_{\lambda_0} \quad (\lambda_0 \in \Lambda)$

定理 B.2. 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$,$\{G_\lambda \mid \lambda \in \Lambda\}$ に対し，次が成立する．

1. $\bigcap_{\lambda\in\Lambda} F_\lambda \subseteq \bigcap_{\lambda\in\Lambda} G_\lambda \leftrightarrow \forall\lambda \in \Lambda[F_\lambda \subseteq G_\lambda]$

2. $\bigcup_{\lambda\in\Lambda} F_\lambda \subseteq \bigcup_{\lambda\in\Lambda} G_\lambda \leftrightarrow \forall\lambda \in \Lambda[F_\lambda \subseteq G_\lambda]$

定理 B.3. 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ と集合 A に対し，次が成立する．

$$A \cup (\bigcap_{\lambda\in\Lambda} F_\lambda) = \bigcap_{\lambda\in\Lambda} (A \cup F_\lambda)$$

問題 B.4. 集合族 $\{F_\lambda \mid \lambda \in \Lambda\}$ と集合 A に対し，次が成り立つことを示せ．

1. $A \cup (\bigcup_{\lambda\in\Lambda} F_\lambda) = \bigcup_{\lambda\in\Lambda} (A \cup F_\lambda)$

2. $A \cap (\bigcap_{\lambda\in\Lambda} F_\lambda) = \bigcap_{\lambda\in\Lambda} (A \cap F_\lambda)$

3. $A \cap (\bigcup_{\lambda\in\Lambda} F_\lambda) = \bigcup_{\lambda\in\Lambda} (A \cap F_\lambda)$

付録C　写像

C.1　像

命題 C.1. $f : U \longrightarrow V, X \subseteq U$ とする．$x \in X$ ならば $f(x) \in f(X)$ である．

命題 C.2. $f : U \longrightarrow V, X \subseteq U$ とする．$y \in f(X)$ であるための必要十分条件は $\exists x \in X[y = f(x)]$ である．

定理 C.3. $f : U \longrightarrow V$ とする．$A, B \subseteq U$ とするとき，次の関係が成り立つ．

1. $A \subseteq B \to f(A) \subseteq f(B)$

2. $f(A \cup B) = f(A) \cup f(B)$

3. $f(A \cap B) \subseteq f(A) \cap f(B)$

C.2　逆像

命題 C.4. $f : U \longrightarrow V, Y \subseteq V$ とする．$\forall x \in U[x \in f^{-1}(Y) \leftrightarrow f(x) \in Y]$ である．

命題 C.5. $f : U \longrightarrow V, Y \subseteq V$ とする．$f^{-1}(Y) = f^{-1}(Y \cap f(U))$ である．

定理 C.6. $f : U \longrightarrow V$ とする．$C, D \subseteq U$ とするとき，次の関係が成り立つ．

1. $C \subseteq D \to f^{-1}(C) \subseteq f^{-1}(D)$
2. $f^{-1}(C \cup D) = f^{-1}(C) \cup f^{-1}(D)$
3. $f^{-1}(C \cap D) = f^{-1}(C) \cap f^{-1}(D)$

問題 C.7. $f : U \longrightarrow V$ とする．$C, D \subseteq V$ とするとき，次の関係が成り立つことを示せ．

1. $f^{-1}(V - C) = U - f^{-1}(C)$
2. $f(f^{-1}(C)) = C \cap f(U)$
3. $C \cap f(U) \subseteq D \cap f(U) \leftrightarrow f^{-1}(C) \subseteq f^{-1}(D)$

付録D　関係

D.1　同値類

定理 D.1. A 上の同値関係 R の同値類に関して次が成り立つ．

1. $\forall a, a' \in A(aRa' \to [a] = [a'])$
2. $\forall a, a' \in A(\neg(aRa') \to [a] \cap [a'] = \varnothing)$
3. $\bigcup_{a \in A} [a] = A$

問題 D.2. 定理 C.1 の 1. の逆を示しなさい．