Chapter 1

Library eo

```
Require Import Arith.
Require Import Ascii.
Require Import Bool.
Require Import Coq.Strings.Byte.
Import IfNotations.

Inductive string : Set :=
  | EmptyString : string
  | String : ascii → string → string.

Inductive EOExpr : Type :=
  | EOUnit : EOExpr
  | EOInt : nat → EOExpr
  | EOBool : bool → EOExpr
  | EOVar : string → EOExpr
  | EOAssign : EOExpr → EOExpr → EOExpr
  | EOPlus : EOExpr → EOExpr → EOExpr
  | EOMinus : EOExpr → EOExpr → EOExpr
  | EOMult : EOExpr → EOExpr → EOExpr
  | EODiv : EOExpr → EOExpr → EOExpr
  | EOIf : EOExpr → EOExpr → EOExpr → EOExpr
  | EOLambda : string → EOExpr → EOExpr
  | EOClosure : string → EOExpr → EOEnv → EOExpr
  | EOApply : EOExpr → EOExpr → EOExpr

with EOEnv : Type :=
  | EOEmptyEnv : EOEnv
  | EOBind : string → EOExpr → EOEnv → EOEnv.
```

Chapter 2

Library main

```
Require Import Bool.

Inductive bool :=
| true
| false.

Definition negb (b : bool) : bool :=
  match b with
  | true => false
  | false => true
  end.

Theorem negb_negb: ∀ (b : bool), negb (negb b) = b.
Proof.
  intros b.
  destruct b.
  + simpl. reflexivity.
  + simpl. reflexivity.
Qed.

Theorem negb_trice : ∀ (b : bool), negb (negb (negb b)) = negb b.
Proof.
  intros b.
  destruct b.
  + simpl. reflexivity.
  + simpl. reflexivity.
Qed.

Definition andb (a b : bool) : bool :=
  match a, b with
  | true, true => true
  | _, _ => false
  end.
```

```
  Theorem and_true_both_arg_true : ∀ (a b : bool),
  a = true → b = true → andb a b = true.
Proof.
  intros a b Ha Hb.
  rewrite Ha.
  rewrite Hb.
  simpl. reflexivity.
Qed.

Theorem and_true_otherside : ∀ (a b : bool),
  andb a b = true → a = true ∧ b = true.
Proof.
  intros a b Ha.
  destruct a.
  destruct b.
  split. reflexivity.
  reflexivity.
  split. reflexivity.
  simpl in Ha.
  inversion Ha.
  destruct b.
  simpl in Ha.
  inversion Ha.
  simpl in Ha.
  inversion Ha.
Qed.

Theorem andb_associative : ∀ (a b c : bool),
  andb a (andb b c) = andb (andb a b) c.
Proof.
  intros [||] [||] [||];
  reflexivity.
Qed.

Theorem andb_comutative : ∀ a b, andb a b = andb b a.
Proof.
  intros [||] [||]; simpl; reflexivity.
Qed.

Definition orb(a b : bool) : bool :=
  match a, b with
  | false, false ⇒ false
  | _, _ ⇒ true
  end.

Theorem andb_negb_orb : ∀ (a b : bool),
```

```
  negb (andb a b) = orb (negb a) (negb b).
Proof.
  intros [||] [||]; simpl; reflexivity.
Qed.

Definition material(a b : bool) : bool :=
match a,b with
  | false, true ⇒ false
  | _, _ ⇒ true
end.

Theorem positive_material : ∀ a b : bool,
  a = true → material a b = true.
  Proof.
    intros a b Ha.
    destruct a eqn:E.
    - simpl. destruct b; reflexivity.
    - discriminate Ha.
    Show Proof.
Qed.
```

Chapter 3

Library phi

```
Require Import Relations.Relation_Definitions.

Inductive Process :=
  | Nil : Process
  | Input : nat → (nat → Process) → Process
  | Output : nat → Process → Process
  | Sum : Process → Process → Process
  | Res : nat → Process → Process.

Definition small_step : relation Process :=
  fun p1 p2 ⇒
    match p1 with
    | Input x f ⇒
      ∃ v, p2 = f v
    | Output x p ⇒
      p2 = p
    | Sum p1' p2' ⇒
      p2 = Sum p1' p2' ∨ p2 = Sum p1 p2'
    | Res x p ⇒
      ∃ y, p2 = Res y p ∧ x ≠ y
    | _ ⇒ False
    end.

Inductive multi_step : relation Process :=
  | refl : ∀ p, multi_step p p
  | step : ∀ p1 p2 p3, small_step p1 p2 → multi_step p2 p3 → multi_step p1 p3.

Inductive bisim : relation Process :=
  | bisim_refl : ∀ p, bisim p p
  | bisim_trans : ∀ p1 p2 p3, bisim p1 p2 → small_step p2 p3 → bisim p1 p3
  | bisim_symm : ∀ p1 p2, bisim p1 p2 → bisim p2 p1.

Definition bisimulation (R : relation Process) :=
  ∀ p q, R p q → (∀ p', small_step p p' → ∃ q', R p' q' ∧ small_step q q') ∧
```

(∀ q', small_step q q' → ∃ p', R p' q' ∧ small_step p p').

`Definition` bisimilarity (R : relation **Process**) :=
∀ p q, R p q → ∃ q', **bisim** q q' ∧ **multi_step** p q'.