Logic

An Idris port of Coq.Init.Logic

Eric Bailey
https://github.com/yurrriq
eric@ericb.me

ABSTRACT

Here I present an Idris port of the `Coq.Init.Logic` module from the Coq standard library.

Keywords

logic, coq, idris

```
||| An Idris port of Coq.Init.Logic
module Logic

import Data.Bifunctor

%access export
```

1. PROPOSITIONAL CONNECTIVES

1.1 Unit

`()` is the always true proposition ($\top$).

```
%elim data Unit = MkUnit
```

1.2 Void

`Void` is the always false proposition ($\perp$).

```
%elim data Void : Type where
```

1.3 Negation

`Not a`, written `~a`, is the negation of `a`.

```
syntax "~" [x] = (Not x)
```

```
Not : Type -> Type
Not a = a -> Void
```

1.4 Conjunction

`And a b`, written `(a, b)`, is the conjunction of `a` and `b`.

`Conj p q` is a proof of `(a, b)` as soon as `p` is a proof of `a` and `q` a proof of `b`.

`proj1` and `proj2` are first and second projections of a conjunction.

```
syntax "(" [a] "," [b] ")" = (And a b)

||| The conjunction of `a` and `b`.
data And : Type -> Type -> Type where
     Conj : a -> b -> (a, b)

implementation Bifunctor And where
    bimap f g (Conj a b) = Conj (f a) (g b)

||| First projection of a conjunction.
proj1 : (a, b) -> a
proj1 (Conj a _) = a

||| Second projection of a conjunction.
proj2 : (a, b) -> b
proj2 (Conj _ b) = b
```

1.5 Disjunction

`Either a b` is the disjunction of `a` and `b`.

```
data Either : Type -> Type -> Type where
     Left  : a -> Either a b
     Right : b -> Either a b
```

1.6 Biconditional

Proof Wiki

$\varphi \vdash \psi$
$\psi \vdash \varphi$
$\overline{\varphi \leftrightarrow \psi}$

`iff a b`, written `a <-> b`, expresses the equivalence of `a` and `b`.

```
infixl 9 <->

||| The biconditional is a *binary connective* that
||| can be voiced: *p* **if and only if** *q*.
public export
(<->) : Type -> Type -> Type
(<->) a b = (a -> b, b -> a)
```

1.6.1 Biconditional is Reflexive

Proof Wiki

$\vdash \varphi \leftrightarrow \varphi$

```
||| The biconditional operator is reflexive.
iffRefl : a <-> a
iffRefl = Conj id id
```

1.6.2 Biconditional is Transitive

Proof Wiki

$$\frac{\varphi \leftrightarrow \psi \qquad \psi \leftrightarrow \chi}{\vdash \varphi \leftrightarrow \chi}$$

```
||| The biconditional operator is transitive.
iffTrans : (a <-> b) -> (b <-> c) -> (a <-> c)
iffTrans (Conj ab ba) (Conj bc cb) =
    Conj (bc . ab) (ba . cb)
```

1.6.3 Biconditional is Commutative

Proof Wiki

$\varphi \leftrightarrow \psi \dashv\vdash \psi \leftrightarrow \varphi$

or

$\vdash (\varphi \leftrightarrow \psi) \leftrightarrow (\psi \leftrightarrow \varphi)$

```
||| The biconditional operator is commutative.
iffSym : (a <-> b) -> (b <-> a)
iffSym (Conj ab ba) = Conj ba ab
```

1.6.4 andIffCompatLeft

$\psi \leftrightarrow \chi \dashv\vdash (\varphi \land \psi) \leftrightarrow (\varphi \land \chi)$

```
andIffCompatLeft : (b <-> c) -> ((a, b) <-> (a, c))
andIffCompatLeft = bimap second second
```

1.6.5 andIffCompatRight

$\psi \leftrightarrow \chi \dashv\vdash (\psi \land \varphi) \leftrightarrow (\chi \land \varphi)$

```
andIffCompatRight : (b <-> c) -> ((b, a) <-> (c, a))
andIffCompatRight = bimap first first
```

1.6.6 orIffCompatLeft

$\psi \leftrightarrow \chi \vdash (\varphi \lor \psi) \leftrightarrow (\varphi \lor \chi)$

```
orIffCompatLeft : (b <-> c) ->
                  (Either a b <-> Either a c)
orIffCompatLeft = bimap second second
```

1.6.7 orIffCompatRight

$\psi \leftrightarrow \chi \vdash (\psi \lor \varphi) \leftrightarrow (\chi \lor \varphi)$

```
orIffCompatRight : (b <-> c) ->
                   (Either b a <-> Either c a)
orIffCompatRight = bimap first first
```

1.6.8 negVoid

$\neg\varphi \dashv\vdash \varphi \leftrightarrow \bot$

or

$\vdash \neg\varphi \leftrightarrow (\varphi \leftrightarrow \bot)$

```
negVoid : (~a) <-> (a <-> Void)
negVoid = Conj (flip Conj void) proj1
```

1.6.9 andCancelLeft

$\psi \to \varphi$
$\chi \to \varphi$
$\overline{((\varphi \land \psi) \leftrightarrow (\varphi \land \chi)) \leftrightarrow (\psi \leftrightarrow \chi)}$

```
andCancelLeft : (b -> a) ->
                (c -> a) ->
                (((a, b) <-> (a, c)) <-> (b <-> c))
andCancelLeft ba ca = Conj (bimap f g) andIffCompatLeft
  where
    f h b = proj2 . h $ Conj (ba b) b
    g h c = proj2 . h $ Conj (ca c) c
```

1.6.10 andCancelRight

```
andCancelRight : (b -> a) ->
                 (c -> a) ->
                 (((b, a) <-> (c, a)) <-> (b <-> c))
andCancelRight ba ca = Conj (bimap f g) andIffCompatRight
  where
    f h b = proj1 . h $ Conj b (ba b)
    g h c = proj1 . h $ Conj c (ca c)
```

1.6.11 Conjunction is Commutative

Proof Wiki

Formulation 1. $\varphi \wedge \psi \dashv\vdash \psi \wedge \varphi$

Formulation 2. $\vdash (\varphi \wedge \psi) \leftrightarrow (\psi \wedge \varphi)$

Source.

```
||| Conjunction is commutative.
andComm : (a, b) <-> (b, a)
andComm = Conj swap swap
  where
    swap : (p, q) -> (q, p)
    swap (Conj p q) = Conj q p
```

1.6.12 Conjunction is Associative

Proof Wiki

Formulation 1. $(\varphi \wedge \psi) \wedge \chi \dashv\vdash \varphi \wedge (\psi \wedge \chi)$

Formulation 2. $\vdash ((\varphi \wedge \psi) \wedge \chi) \leftrightarrow (\varphi \wedge (\psi \wedge \chi))$

Source.

```
||| Conjunction is associative.
andAssoc : ((a, b), c) <-> (a, (b, c))
andAssoc = Conj f g
  where
    f abc@(Conj (Conj a b) c) =
        Conj a (first proj2 abc)
    g abc@(Conj a (Conj b c)) =
        Conj (second proj1 abc) c
```

1.6.13 orCancelLeft

$(\psi \to \neg\varphi) \to (\chi \to \neg\varphi) \to (((\varphi \vee \psi) \leftrightarrow (\varphi \vee \chi)) \leftrightarrow (\psi \leftrightarrow \chi))$

```
orCancelLeft : (b -> ~a) ->
               (c -> ~a) ->
               ((Either a b <-> Either a c) <->
                (b <-> c))
orCancelLeft bNotA cNotA =
    Conj (bimap f g) orIffCompatLeft
  where
    f ef b = go (bNotA b) (ef (Right b))
    g eg c = go (cNotA c) (eg (Right c))
    go : (~a) -> Either a b -> b
    go lf = either (void . lf) id
```

1.6.14 orCancelRight

$$\frac{\psi \vdash \neg\varphi \qquad \chi \vdash \neg\varphi}{((\psi \vee \varphi) \leftrightarrow (\chi \vee \varphi)) \leftrightarrow (\psi \leftrightarrow \chi)}$$

```
orCancelRight : (b -> ~a) ->
                (c -> ~a) ->
                ((Either b a <-> Either c a) <->
                 (b <-> c))
orCancelRight bNotA cNotA =
    Conj (bimap f g) orIffCompatRight
  where
    f ef b = go (bNotA b) (ef (Left b))
    g eg c = go (cNotA c) (eg (Left c))
    go : (~p) -> Either q p -> q
    go rf = either id (void . rf)
```

1.6.15 Disjunction is Commutative

Proof Wiki

$(\varphi \vee \psi) \leftrightarrow (\psi \vee \varphi)$

```
||| Disjunction is commutative.
orComm : Either a b <-> Either b a
orComm = Conj mirror mirror
```

1.6.16 Disjunction is Associative

Proof Wiki

$(\varphi \vee \psi) \vee \chi \vdash \varphi \vee (\psi \vee \chi)$

```
||| Disjunction is associative on the left.
orAssocLeft : Either (Either a b) c ->
              Either a (Either b c)
orAssocLeft = either (second Left) (pure . pure)
```

$\varphi \vee (\psi \vee \chi) \vdash (\varphi \vee \psi) \vee \chi$

```
||| Disjunction is associative on the right.
orAssocRight : Either a (Either b c) ->
               Either (Either a b) c
orAssocRight = either (Left . Left) (first Right)
```

Formulation 1. $(\varphi \vee \psi) \vee \chi \dashv\vdash \varphi \vee (\psi \vee \chi)$

Formulation 2. $\vdash ((\varphi \vee \psi) \vee \chi) \leftrightarrow (\varphi \vee (\psi \vee \chi))$

Source.

```
||| Disjunction is associative.
orAssoc : Either (Either a b) c <->
          Either a (Either b c)
orAssoc = Conj orAssocLeft orAssocRight
```

1.6.17 iffAnd

$\varphi \leftrightarrow \psi \vdash (\varphi \to \psi) \land (\psi \to \varphi)$

```
iffAnd : (a <-> b) -> (a -> b, b -> a)
iffAnd = id
```

1.6.18 iffAndTo

$\varphi \leftrightarrow \psi \dashv\vdash (\varphi \to \psi) \land (\psi \to \varphi)$

or

$\vdash (\varphi \leftrightarrow \psi) \leftrightarrow ((\varphi \to \psi) \land (\psi \to \varphi))$

```
iffToAnd : (a <-> b) <-> (a -> b, b -> a)
iffToAnd = Conj id id
```