```
(* What follows is an example of Coq 8.2 code written using the Proof-general interface for Coq based on
emacs. The code consists of three modules u0, u1 and u01. The need for such an organization in this case
arises from the current state of "universe management" in Coq. While there is a mechanism for automatic
universe level assignment to occurrences
of "Type" expression this mechanism does not allow for a possibility of universe polymorphic constructions.
Since to formulate the equivalence axiom properly we need to define paths spaces for types on at least two
universe levels we have to use "manual" universe management. This is done by the creation of two modules u0
and u1 which are identical except for their names. Each starts with a definition UU:=Type which fixes a
universe named UU. When both are imported into the module u01 we find ourselves in the possession of two
universes u0.UU and u1.UU with all the results and definitions of the modules u0, u1 for each of the
universes. When Coq sees a reference to an identifier from u0 or u1 which is used without the explicit use of
"u0." or "u1." as a prefix it assigns to it prefix "u0." since u0 was imported after u1. As a result UU now
refers to u0.UU etc.

In the first four lines of u01 we rename u0.UU into UU0 and u1.UU into UU1. We also use these two universes
as parts of definitions which force Coq to introduce constrains on the corresponding universe variables which
correspond to the conditions that UU0 is both a member and a sub-universe of UU1. After these constrains have
been established any attempt to use these universes which is incompatible with the constraints e.g. to
consider UU1 as a member of UU0 will generate an error message 'Universe inconsistency". *)


Module u0.

Definition UU:=Type.

Inductive paths (X:UU)(x:X): X -> UU := idpath: paths X x x.

Inductive total2 (X:UU) (P:X -> UU) : UU := tpair: forall x:X, forall p: P x, total2 X P.
Definition pr21 (X:UU) (P:X -> UU) (z: total2 X P) : X :=
match z with tpair x p => x end.
Definition pr22 (X:UU) (P:X -> UU) (z: total2 X P) : P (pr21 _ _ z):=
match z with tpair x p => p end.

Definition iscontr (X:UU) : UU := total2 X (fun cntr:X => forall x:X, paths X x cntr).
Definition iscontrpair (X:UU) (cntr: X) (e: forall x:X, paths X x cntr) : iscontr X :=
tpair X (fun cntr:X => forall x:X, paths X x cntr) cntr e.


Definition hfiber (X:UU)(Y:UU)(f:X -> Y)(y:Y) : UU := total2 X (fun pointover:X => paths Y (f pointover) y).
Definition hfiberpair (X:UU)(Y:UU)(f:X -> Y)(y:Y) (x:X) (e: paths Y (f x) y): hfiber _ _ f y :=
tpair X (fun pointover:X => paths Y (f pointover) y) x e.

Definition isweq (X:UU)(Y:UU)(F:X -> Y) : UU := forall y:Y, iscontr (hfiber X Y F y) .

Lemma idisweq (X:UU) : isweq X X (fun t:X => t).
Proof. intros. unfold isweq. intros. assert (y0: hfiber X X (fun t : X => t) y).
apply (tpair X (fun pointover:X => paths X ((fun t:X => t) pointover) y) y (idpath X y)). split with y0.
intros.
destruct y0. destruct x. induction p. induction p0. apply idpath. Defined.

Definition weq (X:UU)(Y:UU) : UU := total2 (X -> Y) (fun f:X->Y => isweq X Y f) .
Definition weqpair (X:UU)(Y:UU)(f:X-> Y)(is: isweq X Y f) : weq X Y :=
tpair (X -> Y) (fun f:X->Y => isweq X Y f) f is.
Definition idweq (X:UU) : weq X X := tpair (X-> X) (fun f:X->X => isweq X X f) (fun x:X => x) ( idisweq X ) .

Definition invmap (X:UU) (Y:UU) (f:X-> Y) (isw: isweq X Y f): Y->X.
Proof. intros. unfold isweq in isw. apply (pr21 _ _ (pr21 _ _ (isw X0))). Defined.

Definition weqfg (X:UU) (Y:UU) (f:X-> Y) (is1: isweq _ _ f): forall t2:Y, paths Y (f ((invmap _ _ f is1) t2))
t2.
Proof. intros. unfold invmap. simpl. unfold isweq in is1. apply (pr22 _ _ (pr21 _ _ (is1 t2))). Defined.

End u0.
```

```
Module u1.

Definition UU:=Type.

Inductive paths (X:UU)(x:X): X -> UU := idpath: paths X x x.

Inductive total2 (X:UU) (P:X -> UU) : UU := tpair: forall x:X, forall p: P x, total2 X P.
Definition pr21 (X:UU) (P:X -> UU) (z: total2 X P) : X :=
match z with tpair x p => x end.
Definition pr22 (X:UU) (P:X -> UU) (z: total2 X P) : P (pr21 _ _ z):=
match z with tpair x p => p end.

Definition iscontr (X:UU) : UU := total2 X (fun cntr:X => forall x:X, paths X x cntr).
Definition iscontrpair (X:UU) (cntr: X) (e: forall x:X, paths X x cntr) : iscontr X :=
tpair X (fun cntr:X => forall x:X, paths X x cntr) cntr e.


Definition hfiber (X:UU)(Y:UU)(f:X -> Y)(y:Y) : UU := total2 X (fun pointover:X => paths Y (f pointover) y).
Definition hfiberpair (X:UU)(Y:UU)(f:X -> Y)(y:Y) (x:X) (e: paths Y (f x) y): hfiber _ _ f y :=
tpair X (fun pointover:X => paths Y (f pointover) y) x e.

Definition isweq (X:UU)(Y:UU)(F:X -> Y) : UU := forall y:Y, iscontr (hfiber X Y F y) .

Lemma idisweq (X:UU) : isweq X X (fun t:X => t).
Proof. intros. unfold isweq. intros. assert (y0: hfiber X X (fun t : X => t) y).
apply (tpair X (fun pointover:X => paths X ((fun t:X => t) pointover) y) y (idpath X y)). split with y0.
intros.
destruct y0. destruct x. induction p. induction p0. apply idpath. Defined.

Definition weq (X:UU)(Y:UU) : UU := total2 (X -> Y) (fun f:X->Y => isweq X Y f) .
Definition weqpair (X:UU)(Y:UU)(f:X-> Y)(is: isweq X Y f) : weq X Y :=
tpair (X -> Y) (fun f:X->Y => isweq X Y f) f is.
Definition idweq (X:UU) : weq X X := tpair (X-> X) (fun f:X->X => isweq X X f) (fun x:X => x) ( idisweq X ) .

Definition invmap (X:UU) (Y:UU) (f:X-> Y) (isw: isweq X Y f): Y->X.
Proof. intros. unfold isweq in isw. apply (pr21 _ _ (pr21 _ _ (isw X0))). Defined.

Definition weqfg (X:UU) (Y:UU) (f:X-> Y) (is1: isweq _ _ f): forall t2:Y, paths Y (f ((invmap _ _ f is1) t2))
t2.
Proof. intros. unfold invmap. simpl. unfold isweq in is1. apply (pr22 _ _ (pr21 _ _ (is1 t2))). Defined.

End u1.


Module u01.
Import u1 u0.

Definition j01:UU -> u1.UU:= fun T:UU => T.
Definition j11:u1.UU -> u1.UU:=fun T:u1.UU => T.

Definition UU0:=j11 UU.
Definition UU1:=u1.UU.

Definition eqweqmap (X:UU0) (Y:UU0) : (u1.paths _ X Y) -> (weq X Y).
Proof. intros. induction X0. apply idweq. Defined.

Axiom univalenceaxiom: forall X:UU0, forall Y:UU0, u1.isweq (u1.paths Type X Y) (weq X Y) (eqweqmap X Y).

Definition weqtopaths (X:UU0)(Y:UU0)(f:X -> Y)(is:isweq _ _ f): u1.paths _ X Y :=
u1.invmap _ _ (eqweqmap X Y) (univalenceaxiom X Y) (weqpair _ _ f is).

Definition weqpathsweq (X:UU0)(Y:UU0)(f:X -> Y)(is:isweq _ _ f): u1.paths _ (eqweqmap _ _ (weqtopaths _ _ f
is))
(weqpair _ _ f is) := u1.weqfg _ _ (eqweqmap X Y) (univalenceaxiom X Y) (weqpair _ _ f is).

End u01.
```

```
Inductive empty:UU := .
Inductive unit:UU := tt:unit.
Inductive bool:UU := true:bool | false:bool.
Inductive nat:UU := O:nat | S: nat -> nat.


Fixpoint isofhlevel (n:nat) (X:UU): UU:=
match n with
O => iscontr X |
S m => forall x:X, forall x':X, (isofhlevel m (paths _ x x'))
end.

Theorem hlevelsincl (n:nat) (T:UU) : isofhlevel n T -> isofhlevel (S n) T.



Definition isaprop (X:UU): UU := isofhlevel (S O) X.

Theorem isapropempty: isaprop empty.

Theorem isapropunit: isaprop unit.

Theorem isapropiscontr (X:UU0): isaprop (iscontr X).

Theorem isapropisweq (X:UU0)(Y:UU0)(f:X-> Y) : isaprop (isweq _ _ f).

Theorem isapropisofhlevel (n:nat)(X:UU0): isaprop (isofhlevel n X).



Definition isaset (X:UU): UU := isofhlevel (S (S O)) X.

Theorem impred (n:nat)(T:UU0)(P:T -> UU0): (forall t:T, isofhlevel n (P t)) -> (isofhlevel n (forall t:T, P
t)).

Theorem isasetifdec (X:UU): (forall (x x':X), coprod (paths _ x x') (paths _ x x' -> empty)) -> isaset X.

Theorem isasetbool: isaset bool.

Theorem isasetnat: isaset nat.



Definition isofhlevelf (n:nat)(X:UU)(Y:UU)(f:X -> Y): UU := forall y:Y, isofhlevel n (hfiber _ _ f y).
```