```
(*****************************************************************************************************************************)
(* WeakFunctorCategory:                                                                                                      *)
(*                                                                                                                           *)
(*   A category whose morphisms are functors, identified up to natural isomorphism (not equality).  This pulls most of the      *)
(*   heavy lifting out of ReificationsEquivalentToGeneralizedArrows, since the definitions in that context cause Coq to bog     *)
(*   down and run unbearably slowly                                                                                          *)
(*                                                                                                                           *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Categories_ch1_3.
Require Import Functors_ch1_4.
Require Import Isomorphisms_ch1_5.
Require Import ProductCategories_ch1_6_1.
Require Import OppositeCategories_ch1_6_2.
Require Import Enrichment_ch2_8.
Require Import Subcategories_ch7_1.
Require Import NaturalTransformations_ch7_4.
Require Import NaturalIsomorphisms_ch7_5.
Require Import MonoidalCategories_ch7_8.
Require Import Coherence_ch7_8.

Section WeakFunctorCategory.

  (* We can't handle categories directly due to size issues.
   * Therefore, we ask the user to supply two types "Cat" and "Mor"
   * which index the "small categories"; we then construct a large
   * category relative to those. *)
  Structure SmallCategories :=
  { small_cat       : Type
  ; small_ob        : small_cat -> Type
  ; small_hom       : forall c:small_cat, small_ob c -> small_ob c -> Type
  ; small_cat_cat   : forall c:small_cat, Category (small_ob c) (small_hom c)
  }.

  Context {sc:SmallCategories}.
  Structure SmallFunctors :=
  { small_func       : small_cat sc -> small_cat sc -> Type
  ; small_func_fobj  : forall {c1}{c2}, small_func c1 c2 -> (small_ob sc c1 -> small_ob sc c2)
```

```
; small_func_func  : forall {c1}{c2}(f:small_func c1 c2), Functor (small_cat_cat sc c1) (small_cat_cat sc c2) (small_func_fobj f)

(* proof that our chosen indexing contains identity functors and is closed under composition *)
; small_func_id    : forall  c1 , small_func c1 c1
; small_func_id_id : forall {c1}, small_func_func (small_func_id c1) ≃ functor_id (small_cat_cat sc c1)
; small_func_comp  : forall {c1}{c2}{c3}, small_func c1 c2 -> small_func c2 c3 -> small_func c1 c3
; small_func_comp_comp : forall {c1}{c2}{c3}(f:small_func c1 c2)(g:small_func c2 c3),
  small_func_func (small_func_comp f g) ≃ small_func_func f >>>> small_func_func g
}.

Instance WeakFunctorCategory `(sf:SmallFunctors) : Category (small_cat sc) (small_func sf) :=
{ id   := fun a         => small_func_id sf a
; comp := fun a b c f g => small_func_comp sf f g
; eqv  := fun a b f g   => small_func_func sf f ≃ small_func_func sf g
}.
  intros; simpl.
  apply Build_Equivalence.
    unfold Reflexive; simpl; intros; apply if_id.
    unfold Symmetric; simpl; intros; apply if_inv; auto.
    unfold Transitive; simpl; intros; eapply if_comp. apply H. apply H0.
  intros; simpl.
    unfold Proper; unfold respectful; simpl; intros.
    eapply if_comp.
    apply small_func_comp_comp.
    eapply if_inv.
    eapply if_comp.
    apply small_func_comp_comp.
    eapply if_respects. apply if_inv. apply H. apply if_inv. apply H0.
  intros; simpl.
    eapply if_comp.
    apply small_func_comp_comp.
    eapply if_comp; [ idtac | apply if_left_identity ].
    eapply if_respects; try apply if_id.
    apply small_func_id_id.
  intros; simpl.
    eapply if_comp.
    apply small_func_comp_comp.
    eapply if_comp; [ idtac | apply if_right_identity ].
    eapply if_respects; try apply if_id.
    apply small_func_id_id.
  intros; simpl.
```

```
      eapply if_comp.
      eapply if_comp ; [ idtac | apply small_func_comp_comp ].
      apply if_id.
      apply if_inv.
      eapply if_comp.
      eapply if_comp ; [ idtac | apply small_func_comp_comp ].
      apply if_id.
      eapply if_comp.
      eapply if_respects.
      eapply if_id.
      eapply small_func_comp_comp.
      apply if_inv.
      eapply if_comp.
      eapply if_respects.
      eapply small_func_comp_comp.
      eapply if_id.
      set (@if_associativity) as q.
      apply (q _ _ _ _ _ _ _ _ _ _ _ _ _ (small_func_func sf f) _ (small_func_func sf g) _ (small_func_func sf h)).
      Defined.
End WeakFunctorCategory.
Coercion WeakFunctorCategory : SmallFunctors >-> Category.
Coercion small_func_func : small_func >-> Functor.
Coercion small_cat_cat : small_cat >-> Category.
Coercion small_cat : SmallCategories >-> Sortclass.
```