```
(*****************************************************************************************************************************)
(* ReificationCategory:                                                                                                      *)
(*                                                                                                                           *)
(*   There is a category whose objects are surjective monic monoidal enrichments (SMME's) and whose morphisms                *)
(*   are reifications                                                                                                        *)
(*                                                                                                                           *)
(*****************************************************************************************************************************)

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 PreMonoidalCategories.
Require Import MonoidalCategories_ch7_8.
Require Import Coherence_ch7_8.
Require Import Enrichment_ch2_8.
Require Import Enrichments.
Require Import RepresentableStructure_ch7_2.
Require Import Reification.
Require Import WeakFunctorCategory.

(*
 * Technically reifications form merely a *semicategory* (no identity
 * maps), but one can always freely adjoin identity maps (and nothing
 * else) to a semicategory to get a category whose non-identity-map
 * portion is identical to the original semicategory (closing under
 * composition after putting in the identity maps never produces any
 * additional maps)
 *
 * Also, technically this category has ALL enrichments (not just the
 * surjective monic monoidal ones), though there maps OUT OF only the
 * surjective enrichments and INTO only the monic monoidal
 * enrichments.  It's a big pain to do this in Coq, but sort of might
```

```
 * matter in real life: a language with really severe substructural
 * restrictions might fail to be monoidally enriched, meaning we can't
 * use it as a host language.  But that's for the next paper...
 *)
Inductive ReificationOrIdentity : SMMEs -> SMMEs -> Type :=
  | roi_id   : forall smme:SMMEs,                                    ReificationOrIdentity smme smme
  | roi_reif : forall s1 s2:SMMEs, Reification s1 s2 (enr_c_i s2) -> ReificationOrIdentity s1   s2.

Definition reificationOrIdentityFobj s1 s2 (f:ReificationOrIdentity s1 s2) : s1 -> s2 :=
  match f with
  | roi_id   s       => (fun x => x)
  | roi_reif s1 s2 f => reification_rstar_obj f
  end.

Definition reificationOrIdentityFunc
  : forall s1 s2 (f:ReificationOrIdentity s1 s2), Functor (enr_v s1) (enr_v s2) (reificationOrIdentityFobj s1 s2 f).
  intros.
  destruct f.
  apply functor_id.
  unfold reificationOrIdentityFobj.
  apply (reification_rstar_f r).
  Defined.

Definition compose_reifications (s0 s1 s2:SMMEs) :
  Reification s0 s1 (enr_c_i s1) -> Reification s1 s2 (enr_c_i s2) -> Reification s0 s2 (enr_c_i s2).
  intros.
  refine
    {| reification_rstar_f := reification_rstar_f X >>>> reification_rstar_f X0
     ; reification_rstar   := PreMonoidalFunctorsCompose (reification_rstar X) (reification_rstar X0)
     ; reification_r       := fun K => (reification_r X K) >>>> (reification_r X0 (enr_c_i s1))
    |}.
  intro K.
  set (ni_associativity (reification_r X K) (reification_r X0 (enr_c_i s1)) (HomFunctor s2 (enr_c_i s2))) as q.
  eapply ni_comp.
  apply q.
  clear q.
  set (reification_commutes X K) as comm1.
  set (reification_commutes X0 (enr_c_i s1)) as comm2.
  set (HomFunctor s0 K) as a in *.
  set (reification_rstar_f X) as a' in *.
  set (reification_rstar_f X0) as x in *.
```

```
  set (reification_r X K) as b in *.
  set (reification_r X0 (enr_c_i s1)) as c in *.
  set (HomFunctor s2 (enr_c_i s2)) as c' in *.
  set (HomFunctor s1 (enr_c_i s1)) as b' in *.
  apply (ni_comp(F2:=b >>>> (b' >>>> x))).
  apply (ni_respects1 b (c >>>> c') (b' >>>> x)).
  apply comm2.
  eapply ni_comp.
  eapply ni_inv.
  apply (ni_associativity b b' x).
  eapply ni_inv.
  eapply ni_comp.
  eapply ni_inv.
  apply (ni_associativity a a' x).
  apply (ni_respects2 (a >>>> a') (b >>>> b') x).
  apply ni_inv.
  apply comm1.
  apply (reification_host_lang_pure _ _ _ X0).
  Defined.

Definition reificationOrIdentityComp
  : forall s1 s2 s3, ReificationOrIdentity s1 s2 -> ReificationOrIdentity s2 s3 -> ReificationOrIdentity s1 s3.
  intros.
  destruct X.
    apply X0.
  destruct X0.
    apply (roi_reif _ _ r).
    apply (roi_reif _ _ (compose_reifications _ _ _ r r0)).
    Defined.

Definition MorphismsOfCategoryOfReifications : @SmallFunctors SMMEs.
  refine {| small_func      := ReificationOrIdentity
          ; small_func_id   := fun s => roi_id s
          ; small_func_func := fun smme1 smme2 f => reificationOrIdentityFunc _ _ f
          ; small_func_comp := reificationOrIdentityComp
         |}; intros; simpl.
  apply if_id.
  destruct f as [| fobj f]; simpl in *.
    apply if_inv.
    apply if_left_identity.
  destruct g as [| gobj g]; simpl in *.
```

```
    apply if_inv.
    apply if_right_identity.
  apply if_id.
  Defined.

Definition CategoryOfReifications :=
  WeakFunctorCategory MorphismsOfCategoryOfReifications.
```