```
(*****************************************************************************************************************************)
(* ProgrammingLanguage                                                                                                       *)
(*                                                                                                                           *)
(*   Basic assumptions about programming languages.                                                                          *)
(*                                                                                                                           *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Categories_ch1_3.
Require Import InitialTerminal_ch2_2.
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 BinoidalCategories.
Require Import PreMonoidalCategories.
Require Import MonoidalCategories_ch7_8.
Require Import Coherence_ch7_8.
Require Import Enrichment_ch2_8.
Require Import RepresentableStructure_ch7_2.
Require Import FunctorCategories_ch7_7.

Require Import Enrichments.
Require Import NaturalDeduction.
Require Import NaturalDeductionCategory.

Section Programming_Language.

  Context {T    : Type}.               (* types of the language *)

  Definition PLJudg := (Tree ??T) * (Tree ??T).
  Definition sequent := @pair (Tree ??T) (Tree ??T).
     Notation "cs |= ss" := (sequent cs ss) : pl_scope.

  Context {Rule : Tree ??PLJudg -> Tree ??PLJudg -> Type}.
```

```
Notation "H /··/· C" := (ND Rule H C) : pl_scope.

Open Scope pf_scope.
Open Scope nd_scope.
Open Scope pl_scope.

Class ProgrammingLanguage :=
{ pl_eqv0               :  @ND_Relation PLJudg Rule
; pl_snd                :> @SequentND PLJudg Rule _ sequent
; pl_cnd                :> @ContextND PLJudg Rule T sequent pl_snd
; pl_eqv1               :> @SequentND_Relation PLJudg Rule _ sequent pl_snd pl_eqv0
; pl_eqv                :> @ContextND_Relation PLJudg Rule _ sequent pl_snd pl_cnd pl_eqv0 pl_eqv1
}.
Notation "pf1 === pf2" := (@ndr_eqv _ _ pl_eqv _ _ pf1 pf2) : temporary_scope3.

Section LanguageCategory.

  Context (PL:ProgrammingLanguage).

  (* category of judgments in a fixed type/coercion context *)
  Definition Judgments_cartesian := @Judgments_Category_CartesianCat _ Rule pl_eqv.

  Definition JudgmentsL          := Judgments_cartesian.

  Definition identityProof t : [] ~~{JudgmentsL}~~> [t |= t].
    unfold hom; simpl.
    apply snd_initial.
    Defined.

  Definition cutProof a b c : [a |= b],,[b |= c] ~~{JudgmentsL}~~> [a |= c].
    unfold hom; simpl.
    apply snd_cut.
    Defined.

  Existing Instance pl_eqv.

  Definition TypesL : ECategory JudgmentsL (Tree ??T) (fun x y => [x|=y]).
    refine
    {| eid   := identityProof
     ; ecomp := cutProof
```

```
  |}; intros.
  apply (mon_commutative(MonoidalCat:=JudgmentsL)).
  apply (mon_commutative(MonoidalCat:=JudgmentsL)).
  unfold identityProof; unfold cutProof; simpl; eapply cndr_inert. apply pl_eqv. auto. auto.
  unfold identityProof; unfold cutProof; simpl; eapply cndr_inert. apply pl_eqv. auto. auto.
  unfold identityProof; unfold cutProof; simpl; eapply cndr_inert. apply pl_eqv. auto. auto.
  apply ndpc_comp; auto.
  apply ndpc_comp; auto.
  Defined.

Instance Types_first c : EFunctor TypesL TypesL (fun x => x,,c ) :=
  { efunc := fun x y => cnd_expand_right(ContextND:=pl_cnd) x y c }.
  intros; apply (mon_commutative(MonoidalCat:=JudgmentsL)).
  intros. unfold ehom. unfold hom. unfold identityProof. unfold eid. simpl. unfold identityProof.
  apply (cndr_inert pl_cnd); auto.
  intros. unfold ehom. unfold comp. simpl. unfold cutProof.
  rewrite <- (@ndr_prod_preserves_comp _ _ pl_eqv _ _ (cnd_expand_right _ _ c) _ _ (nd_id1 (b|=c0))
              _ (nd_id1 (a,,c |= b,,c))  _ (cnd_expand_right _ _ c)).
  setoid_rewrite (@ndr_comp_right_identity _ _ pl_eqv _ [a,, c |= b,, c]).
  setoid_rewrite (@ndr_comp_left_identity  _ _ pl_eqv [b |= c0]).
  simpl; eapply cndr_inert. apply pl_eqv. auto. auto.
  Defined.

Instance Types_second c : EFunctor TypesL TypesL (fun x => c,,x) :=
  { efunc := fun x y => ((@cnd_expand_left _ _ _ _ _ _ x y c)) }.
  intros; apply (mon_commutative(MonoidalCat:=JudgmentsL)).
  intros. unfold ehom. unfold hom. unfold identityProof. unfold eid. simpl. unfold identityProof.
  eapply cndr_inert; auto. apply pl_eqv.
  intros. unfold ehom. unfold comp. simpl. unfold cutProof.
  rewrite <- (@ndr_prod_preserves_comp _ _ pl_eqv _ _ (cnd_expand_left _ _ c) _ _ (nd_id1 (b|=c0))
              _ (nd_id1 (c,,a |= c,,b))  _ (cnd_expand_left _ _ c)).
  setoid_rewrite (@ndr_comp_right_identity _ _ pl_eqv _ [c,,a |= c,,b]).
  setoid_rewrite (@ndr_comp_left_identity  _ _ pl_eqv [b |= c0]).
  simpl; eapply cndr_inert. apply pl_eqv. auto. auto.
  Defined.

Definition Types_binoidal : EBinoidalCat TypesL (@T_Branch _).
  refine
    {| ebc_first  := Types_first
     ; ebc_second := Types_second
     |}.
```

```
  Defined.

Instance Types_assoc_iso a b c : Isomorphic(C:=TypesL) ((a,,b),,c) (a,,(b,,c)) :=
  { iso_forward  := snd_initial _ ;; cnd_ant_cossa _ a b c
  ; iso_backward := snd_initial _ ;; cnd_ant_assoc _ a b c
  }.
  simpl; eapply cndr_inert. unfold identityProof; apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
  simpl; eapply cndr_inert. unfold identityProof; apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
    Defined.

Instance Types_cancelr_iso a : Isomorphic(C:=TypesL) (a,,[]) a :=
  { iso_forward  := snd_initial _ ;; cnd_ant_rlecnac _ a
  ; iso_backward := snd_initial _ ;; cnd_ant_cancelr _ a
  }.
  unfold eqv; unfold comp; simpl.
  eapply cndr_inert. apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
  unfold eqv; unfold comp; simpl.
  eapply cndr_inert. apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
  Defined.

Instance Types_cancell_iso a : Isomorphic(C:=TypesL) ([],,a) a :=
  { iso_forward  := snd_initial _ ;; cnd_ant_llecnac _ a
  ; iso_backward := snd_initial _ ;; cnd_ant_cancell _ a
  }.
  unfold eqv; unfold comp; simpl.
  eapply cndr_inert. apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
```

```
  unfold eqv; unfold comp; simpl.
  eapply cndr_inert. apply pl_eqv. auto.
    apply ndpc_comp; auto.
    apply ndpc_comp; auto.
    auto.
  Defined.

Instance Types_assoc a b : Types_second a >>>> Types_first b <~~~> Types_first b >>>> Types_second a :=
  { ni_iso := fun c => Types_assoc_iso a c b }.
  admit.   (* need to add this as an obligation in ProgrammingLanguage class *)
  Defined.

Instance Types_cancelr   : Types_first [] <~~~> functor_id _ :=
  { ni_iso := Types_cancelr_iso }.
  intros; simpl.
  admit.   (* need to add this as an obligation in ProgrammingLanguage class *)
  Defined.

Instance Types_cancell   : Types_second [] <~~~> functor_id _ :=
  { ni_iso := Types_cancell_iso }.
  admit.   (* need to add this as an obligation in ProgrammingLanguage class *)
  Defined.

Instance Types_assoc_ll a b : Types_second (a,,b) <~~~> Types_second b >>>> Types_second a :=
  { ni_iso := fun c => Types_assoc_iso a b c }.
  admit.   (* need to add this as an obligation in ProgrammingLanguage class *)
  Defined.

Instance Types_assoc_rr a b : Types_first (a,,b) <~~~> Types_first a >>>> Types_first b :=
  { ni_iso := fun c => iso_inv _ _ (Types_assoc_iso c a b) }.
  admit.   (* need to add this as an obligation in ProgrammingLanguage class *)
  Defined.

Instance TypesL_PreMonoidal : PreMonoidalCat Types_binoidal [] :=
  { pmon_assoc    := Types_assoc
  ; pmon_cancell  := Types_cancell
  ; pmon_cancelr  := Types_cancelr
  ; pmon_assoc_rr := Types_assoc_rr
  ; pmon_assoc_ll := Types_assoc_ll
  }.
  apply Build_Pentagon.
```

```
  intros; simpl.
  eapply cndr_inert. apply pl_eqv.
  apply ndpc_comp.
  apply ndpc_comp.
  auto.
  apply ndpc_comp.
  apply ndpc_prod.
  apply ndpc_comp.
  apply ndpc_comp.
  auto.
  apply ndpc_comp.
  auto.
  auto.
  auto.
  auto.
  auto.
  auto.
  apply ndpc_comp.
  apply ndpc_comp.
  auto.
  apply ndpc_comp.
  auto.
  auto.
  auto.
apply Build_Triangle; intros; simpl.
  eapply cndr_inert. apply pl_eqv.
  auto.
  apply ndpc_comp.
  apply ndpc_comp.
  auto.
  apply ndpc_comp.
  auto.
  auto.
  auto.
  eapply cndr_inert. apply pl_eqv. auto.
    auto.
intros; simpl; reflexivity.
intros; simpl; reflexivity.
admit.  (* assoc   is central: need to add this as an obligation in ProgrammingLanguage class *)
admit.  (* cancelr is central: need to add this as an obligation in ProgrammingLanguage class *)
admit.  (* cancell is central: need to add this as an obligation in ProgrammingLanguage class *)
```

```
    Defined.

  Definition TypesEnrichedInJudgments : SurjectiveEnrichment.
    refine
      {| senr_c_pm     := TypesL_PreMonoidal
       ; senr_v        := JudgmentsL
       ; senr_v_bin    := Judgments_Category_binoidal _
       ; senr_v_pmon   := Judgments_Category_premonoidal _
       ; senr_v_mon    := Judgments_Category_monoidal _
       ; senr_c_bin    := Types_binoidal
       ; senr_c        := TypesL
      |}.
    Defined.

End LanguageCategory.

End Programming_Language.
Implicit Arguments ND [ Judgment ].
```