```
(*********************************************************************************************************************************)
(* NaturalDeductionCategory:                                                                                                     *)
(*                                                                                                                               *)
(*   Natural Deduction proofs form a category                                                                                    *)
(*                                                                                                                               *)
(*********************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import NaturalDeduction.

Require Import Algebras_ch4.
Require Import Categories_ch1_3.
Require Import Functors_ch1_4.
Require Import Isomorphisms_ch1_5.
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 Coherence_ch7_8.
Require Import InitialTerminal_ch2_2.
Require Import BinoidalCategories.
Require Import PreMonoidalCategories.
Require Import MonoidalCategories_ch7_8.

Open Scope nd_scope.
Open Scope pf_scope.

(* proofs form a category, with judgment-trees as the objects *)
Section Judgments_Category.

  Context {Judgment : Type}.
  Context {Rule     : forall (hypotheses:Tree ??Judgment)(conclusion:Tree ??Judgment), Type}.
  Context (nd_eqv   : @ND_Relation Judgment Rule).

  Notation "pf1 === pf2" := (@ndr_eqv _ _ nd_eqv _ _ pf1 pf2).

  (* there is a category whose objects are judgments and whose morphisms are proofs *)
  Instance Judgments_Category : Category (Tree ??Judgment) (fun h c => h /⋯⋯/ c) :=
{ id   := fun h          => nd_id _
; comp := fun a b c f g  => f ;; g
; eqv  := fun a b f g    => f === g
}.
intros; apply Build_Equivalence;
  [ unfold Reflexive; intros; reflexivity
  | unfold Symmetric; intros; symmetry; auto
  | unfold Transitive; intros; transitivity y; auto ].
unfold Proper; unfold respectful; intros; simpl; apply ndr_comp_respects; auto.
intros; apply (ndr_builtfrom_structural f); auto.
intros; apply (ndr_builtfrom_structural f); auto.
intros; apply ndr_comp_associativity.
Defined.

(* Judgments form a binoidal category *)
Instance jud_first (a:Judgments_Category) : Functor Judgments_Category Judgments_Category (fun x => x,,a) :=
  { fmor := fun b c (f:b /⋯⋯/ c) => f ** (nd_id a) }.
  intros; unfold eqv; simpl; apply ndr_prod_respects; auto.
  intros; unfold eqv in *; simpl in *; reflexivity.
  intros; unfold eqv in *; simpl in *; apply (ndr_builtfrom_structural (nd_id a)); auto.
  intros; unfold eqv in *; simpl in *.
    setoid_rewrite <- ndr_prod_preserves_comp.
    apply (ndr_builtfrom_structural (f;;g)); auto.
  Defined.
Instance jud_second (a:Judgments_Category) : Functor Judgments_Category Judgments_Category (fun x => a,,x) :=
  { fmor := fun b c (f:b /⋯⋯/ c) => (nd_id a) ** f }.
  intros; unfold eqv; simpl; apply ndr_prod_respects; auto.
  intros; unfold eqv in *; simpl in *; reflexivity.
  intros; unfold eqv in *; simpl in *; apply (ndr_builtfrom_structural (nd_id a)); auto.
  intros; unfold eqv in *; simpl in *.
    setoid_rewrite <- ndr_prod_preserves_comp.
    apply (ndr_builtfrom_structural (f;;g)); auto.
  Defined.
Instance Judgments_Category_binoidal : BinoidalCat Judgments_Category (@T_Branch (??Judgment)) :=
  { bin_first  := jud_first
  ; bin_second := jud_second }.

(* and that category is commutative (all morphisms central) *)
Instance Judgments_Category_Commutative : CommutativeCat Judgments_Category_binoidal.
  apply Build_CommutativeCat.
  intros; apply Build_CentralMorphism; intros; unfold eqv; simpl in *.
  setoid_rewrite <- (ndr_prod_preserves_comp (nd_id a) g f (nd_id d)).
    setoid_rewrite <- (ndr_prod_preserves_comp f (nd_id _) (nd_id _) g).
    setoid_rewrite ndr_comp_left_identity.
    setoid_rewrite ndr_comp_right_identity.
    reflexivity.
  setoid_rewrite <- (ndr_prod_preserves_comp (nd_id _) f g (nd_id _)).
    setoid_rewrite <- (ndr_prod_preserves_comp g (nd_id _) (nd_id _) f).
    setoid_rewrite ndr_comp_left_identity.
    setoid_rewrite ndr_comp_right_identity.
    reflexivity.
    Defined.

(* Judgments form a premonoidal category *)
Definition jud_assoc_iso (a b c:Judgments_Category) : @Isomorphic _ _ Judgments_Category ((a,,b),,c) (a,,(b,,c)).
  refine {| iso_forward  := nd_assoc ; iso_backward := nd_cossa |}.
  unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural nd_id0); auto.
  unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural nd_id0); auto.
  Defined.
Definition jud_cancelr_iso (a:Judgments_Category) : @Isomorphic _ _ Judgments_Category (a,,[]) a.
  refine {| iso_forward  := nd_cancelr ; iso_backward := nd_rlecnac |};
  unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural nd_id0); auto.
  Defined.
Definition jud_cancell_iso (a:Judgments_Category) : @Isomorphic _ _ Judgments_Category ([],,a) a.
  refine {| iso_forward  := nd_cancell ; iso_backward := nd_llecnac |};
  unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural nd_id0); auto.
  Defined.
Instance jud_mon_cancelr : jud_first [] <~~~> functor_id Judgments_Category :=
  { ni_iso := jud_cancelr_iso }.
  intros; unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural f); auto.
  Defined.
Instance jud_mon_cancell : jud_second [] <~~~> functor_id Judgments_Category :=
  { ni_iso := jud_cancell_iso }.
  intros; unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural f); auto.
  Defined.
Instance jud_mon_assoc : forall a b, a ⋊- >>>> - ⋈b <~~~> - ⋈b >>>> a ⋊- :=
  { ni_iso := fun c => jud_assoc_iso a c b }.
  intros; unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural f); auto.
  Defined.
Instance jud_mon_assoc_rr : forall a b, - ⋉(a ⊗ b) <~~~> - ⋊a >>>> - ⋉b.
  intros.
  apply ni_inv.
```

```
  refine {| ni_iso := fun c => (jud_assoc_iso _ _ _) |}.
  intros; unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural f); auto.
  Defined.
Instance jud_mon_assoc_ll : forall a b, (a ⊗ b) ⋊ <~~~> b ⋊ >>>> a ⋊ :=
  { ni_iso := fun c => jud_assoc_iso _ _ _ }.
  intros; unfold eqv; unfold comp; simpl; apply (ndr_builtfrom_structural f); auto.
  Defined.
Instance Judgments_Category_premonoidal : PreMonoidalCat Judgments_Category_binoidal [] :=
{ pmon_cancelr  := jud_mon_cancelr
; pmon_cancell  := jud_mon_cancell
; pmon_assoc    := jud_mon_assoc
; pmon_assoc_rr := jud_mon_assoc_rr
; pmon_assoc_ll := jud_mon_assoc_ll
}.
  unfold functor_fobj; unfold fmor; simpl;
    apply Build_Pentagon; simpl; intros; apply (ndr_builtfrom_structural nd_id0); auto.
  unfold functor_fobj; unfold fmor; simpl;
    apply Build_Triangle; simpl; intros; apply (ndr_builtfrom_structural nd_id0); auto.
  intros; unfold eqv; simpl; auto; reflexivity.
  intros; unfold eqv; simpl; auto; reflexivity.
  intros; unfold eqv; simpl; apply Judgments_Category_Commutative.
  intros; unfold eqv; simpl; apply Judgments_Category_Commutative.
  intros; unfold eqv; simpl; apply Judgments_Category_Commutative.
    Defined.

(* commutative premonoidal categories are monoidal *)
Instance Judgments_Category_monoidal : MonoidalCat Judgments_Category_premonoidal :=
  { mon_commutative := Judgments_Category_Commutative }.

(* Judgments also happens to have a terminal object - the empty list of judgments *)
Instance Judgments_Category_Terminal : TerminalObject Judgments_Category [].
  refine {| drop := nd_weak ; drop_unique := _ |}.
    abstract (intros; unfold eqv; simpl; apply ndr_void_proofs_irrelevant).
  Defined.

(* Judgments is also a diagonal category via nd_copy *)
Instance Judgments_Category_Diagonal : DiagonalCat Judgments_Category_monoidal.
  intros.
  refine {| copy := nd_copy |}; intros; simpl.
  setoid_rewrite ndr_comp_associativity.
    setoid_rewrite <- (ndr_prod_preserves_copy f).
```

```
      apply ndr_comp_respects; try reflexivity.
      etransitivity.
      symmetry.
      apply ndr_prod_preserves_comp.
      setoid_rewrite ndr_comp_left_identity.
      setoid_rewrite ndr_comp_right_identity.
      reflexivity.
    setoid_rewrite ndr_comp_associativity.
      setoid_rewrite <- (ndr_prod_preserves_copy f).
      apply ndr_comp_respects; try reflexivity.
      etransitivity.
      symmetry.
      apply ndr_prod_preserves_comp.
      setoid_rewrite ndr_comp_left_identity.
      setoid_rewrite ndr_comp_right_identity.
      reflexivity.
      Defined.

  (* Judgments is a cartesian category: it has a terminal object, diagonal morphisms, and the right naturalities *)
  Instance Judgments_Category_CartesianCat : CartesianCat Judgments_Category_monoidal :=
    { car_terminal := Judgments_Category_Terminal ; car_diagonal := Judgments_Category_Diagonal }.
    intros; unfold eqv; simpl; symmetry; apply ndr_copy_then_weak_left.
    intros; unfold eqv; simpl; symmetry; apply ndr_copy_then_weak_right.
    Defined.

End Judgments_Category.

Close Scope pf_scope.
Close Scope nd_scope.
```