```
(*****************************************************************************************************************************)
(* HaskStrong: a dependently-typed version of CoreSyn                                                                         *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Coq.Strings.String.
Require Import Coq.Lists.List.
Require Import HaskKinds.
Require Import HaskCoreTypes.
Require Import HaskLiteralsAndTyCons.
Require Import HaskStrongTypes.
Require Import HaskWeakVars.
Require Import HaskCoreVars.

Section HaskStrong.

  (* any type with decidable equality may be used to represent value variables *)
  Context `{EQD_VV:EqDecidable VV}.

  (* a StrongCaseBranchWithVVs contains all the data found in a case branch except the expression itself *)
  Record StrongCaseBranchWithVVs {tc:TyCon}{Γ}{atypes:IList _ (HaskType Γ) (tyConKind tc)}{sac:@StrongAltCon tc} :=
  { scbwv_exprvars          :  vec VV (sac_numExprVars sac)
  ; scbwv_exprvars_distinct :  distinct (vec2list scbwv_exprvars)
  ; scbwv_varstypes         := vec_zip scbwv_exprvars (sac_types sac Γ atypes)
  ; scbwv_ξ                 := fun ξ lev =>  update_ξ (weakLT'∘ξ) (weakL' lev) (vec2list scbwv_varstypes)
  }.
  Implicit Arguments StrongCaseBranchWithVVs [[Γ]].

  Inductive Expr : forall Γ (Δ:CoercionEnv Γ), (VV -> LeveledHaskType Γ ★) -> LeveledHaskType Γ ★-> Type :=

  (* an "EGlobal" is any variable which is free in the expression which was passed to -fcoqpass (ie bound outside it) *)
  | EGlobal: ∀ Γ Δ ξ t,       WeakExprVar ->                                                    Expr Γ Δ ξ t

  | EVar   : ∀ Γ Δ ξ ev,                                                                        Expr Γ Δ ξ (ξ ev)
  | ELit   : ∀ Γ Δ ξ lit   l,                                                                   Expr Γ Δ ξ (literalType lit@@l)
  | EApp   : ∀ Γ Δ ξ t1 t2 l,        Expr Γ Δ ξ (t2--->t1 @@ l)   -> Expr Γ Δ ξ (t2 @@ l)        -> Expr Γ Δ ξ (t1 @@ l)
  | ELam   : ∀ Γ Δ ξ t1 t2 l ev,             Expr Γ Δ (update_ξ ξ l ((ev,t1)::nil)) (t2@@l)      -> Expr Γ Δ ξ (t1--->t2@@l)
  | ELet   : ∀ Γ Δ ξ tv t  l ev,Expr Γ Δ ξ (tv@@l)->Expr Γ Δ (update_ξ ξ l ((ev,tv)::nil))(t@@l)  -> Expr Γ Δ ξ (t@@l)
  | EEsc   : ∀ Γ Δ ξ ec t  l,     Expr Γ Δ ξ (<[ ec |- t ]> @@ l)                               -> Expr Γ Δ ξ (t @@ (ec::l))
```

```
| EBrak  : ∀ Γ Δ ξ ec t  l,     Expr Γ Δ ξ (t @@ (ec::l))                                   -> Expr Γ Δ ξ (<[ ec |- t ]> @@ l)
| ECast  : forall Γ Δ ξ t1 t2 (γ:HaskCoercion Γ Δ (t1 ~~~ t2)) l,
  Expr Γ Δ ξ (t1 @@ l)                      -> Expr Γ Δ ξ (t2 @@ l)
| ENote  : ∀ Γ Δ ξ t, Note                    -> Expr Γ Δ ξ t                             -> Expr Γ Δ ξ t
| ETyApp : ∀ Γ Δ κ σ τ ξ l,                   Expr Γ Δ ξ (HaskTAll κ σ @@ l)              -> Expr Γ Δ ξ (substT σ τ @@ l)
| ECoLam : forall Γ Δ κ σ (σ1 σ2:HaskType Γ κ) ξ l,
  Expr Γ (σ1~~~σ2::Δ) ξ (σ @@ l)    -> Expr Γ Δ ξ (σ1~~σ2    ⇒σ @@ l)
| ECoApp : forall Γ Δ κ (σ1 σ2:HaskType Γ κ) (γ:HaskCoercion Γ Δ (σ1~~~σ2)) σ ξ l,
  Expr Γ Δ ξ (σ1 ~~ σ2 ⇒σ @@ l)           -> Expr Γ Δ ξ (σ       @@l)
| ETyLam : ∀ Γ Δ ξ κ σ l,
  Expr (κ::Γ) (weakCE Δ) (weakLT∘ξ) (HaskTApp (weakF σ) (FreshHaskTyVar _)@@(weakL l))-> Expr Γ Δ ξ (HaskTAll κ σ @@ l)
| ECase    : forall Γ Δ ξ l tc tbranches atypes,
             Expr Γ Δ ξ (caseType tc atypes @@ l) ->
             Tree ??{ sac : _
                  & { scb : StrongCaseBranchWithVVs tc atypes sac
                  &         Expr (sac_Γ sac Γ)
                                 (sac_Δ sac Γ atypes (weakCK'' Δ))
                                 (scbwv_ξ scb ξ l)
                                 (weakLT' (tbranches@@l)) } } ->
             Expr Γ Δ ξ (tbranches @@ l)

| ELetRec  : ∀ Γ Δ ξ l τ vars,
  let ξ' := update_ξ ξ l (leaves vars) in
  ELetRecBindings Γ Δ ξ'     l vars ->
  Expr            Γ Δ ξ' (τ@@l) ->
  Expr            Γ Δ ξ  (τ@@l)

(* can't avoid having an additional inductive: it is a tree of Expr's, each of whose ξ depends on the type of the entire tree *)
with ELetRecBindings : ∀ Γ, CoercionEnv Γ -> (VV -> LeveledHaskType Γ ★) -> HaskLevel Γ -> Tree ??(VV*HaskType Γ ★) -> Type :=
| ELR_nil    : ∀ Γ Δ ξ l  ,                                                          ELetRecBindings Γ Δ ξ l []
| ELR_leaf   : ∀ Γ Δ ξ l v t,                                          Expr Γ Δ ξ (t @@ l) -> ELetRecBindings Γ Δ ξ l [(v,t)]
| ELR_branch : ∀ Γ Δ ξ l t1 t2, ELetRecBindings Γ Δ ξ l t1 -> ELetRecBindings Γ Δ ξ l t2 -> ELetRecBindings Γ Δ ξ l (t1,,t2)
.

Context {ToStringVV:ToString VV}.
Context {ToLatexVV:ToLatex VV}.
Fixpoint exprToString {Γ}{Δ}{ξ}{τ}(exp:Expr Γ Δ ξ τ) : string :=
  match exp with
  | EVar  Γ' _ ξ' ev             => "var."+++ toString ev
  | EGlobal Γ' _ ξ' t wev        => "global." +++ toString (wev:CoreVar)
  | ELam  Γ'   _ _ tv _ _ cv e   => "\("+++ toString cv +++":t) -> "+++ exprToString e
```

```
    | ELet  Γ' _ _ t _ _ ev e1 e2   => "let "+++toString ev+++" = "+++exprToString e1+++" in "+++exprToString e2
    | ELit  _ _ _ lit _             => "lit."+++toString lit
    | EApp  Γ' _ _ _ _ _ e1 e2      => "("+++exprToString e1+++")("+++exprToString e2+++")"
    | EEsc  Γ' _ _ ec t _ e         => "~~("+++exprToString e+++")"
    | EBrak Γ' _ _ ec t _ e         => "<["+++exprToString e+++"]>"
    | ENote _ _ _ _ n e             => "note."+++exprToString e
    | ETyApp  Γ Δ κ σ τ ξ l       e => "("+++exprToString e+++")@("+++toString τ+++")"
    | ECoApp Γ Δ κ σ₁ σ₂ γ σ ξ l e  => "("+++exprToString e+++")~(co)"
    | ECast Γ Δ ξ t1 t2 γ l e       => "cast ("+++exprToString e+++"):t2"
    | ETyLam _ _ _ k _ _ e          => "\@_ ->"+++ exprToString e
    | ECoLam Γ Δ κ σ σ₁ σ₂ ξ l e    => "\~_ ->"+++ exprToString e
    | ECase Γ Δ ξ l tc tbranches atypes escrut alts => "case " +++ exprToString escrut +++ " of FIXME"
    | ELetRec _ _ _ _ _ vars elrb e => "letrec FIXME in " +++ exprToString e
    end.
  Instance ExprToString Γ Δ ξ τ : ToString (Expr Γ Δ ξ τ) := { toString := exprToString }.

End HaskStrong.
Implicit Arguments StrongCaseBranchWithVVs [[Γ]].
```