```
(*****************************************************************************************************************************)
(* HaskWeakToCore: convert HaskWeak to HaskCore                                                                              *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Coq.Strings.String.
Require Import Coq.Lists.List.
Require Import HaskKinds.
Require Import HaskLiteralsAndTyCons.
Require Import HaskCoreVars.
Require Import HaskCoreTypes.
Require Import HaskCore.
Require Import HaskWeakVars.
Require Import HaskWeakTypes.
Require Import HaskWeak.
Require Import HaskCoreToWeak.

Variable mkCoreLet : @CoreBind CoreVar -> @CoreExpr CoreVar -> @CoreExpr CoreVar.
  Extract Inlined Constant mkCoreLet => "MkCore.mkCoreLet".

Variable sortAlts  : forall {a}{b}, list (@triple CoreAltCon a b) -> list (@triple CoreAltCon a b).
  Extract Inlined Constant sortAlts => "sortAlts".
  Implicit Arguments sortAlts [[a][b]].

Variable mkUnsafeCoercion : CoreType -> CoreType -> CoreCoercion.
    Extract Inlined Constant mkUnsafeCoercion => "Coercion.mkUnsafeCoercion".

(* Coercion and Type are actually the same thing in GHC, but we don't tell Coq about that.  This lets us get around it. *)
Variable coreCoercionsAreReallyTypes : CoreCoercion -> CoreType.
  Extract Inlined Constant coreCoercionsAreReallyTypes => "(\x -> x)".

Definition weakAltConToCoreAltCon (wa:WeakAltCon) : CoreAltCon :=
  match wa with
  | WeakDataAlt cdc => DataAlt cdc
  | WeakLitAlt  lit => LitAlt lit
  | WeakDEFAULT     => DEFAULT
  end.

Fixpoint weakTypeToCoreType (wt:WeakType) : CoreType :=
```

```
  match wt with
    | WTyVarTy  (weakTypeVar v _)     => TyVarTy v
    | WAppTy (WAppTy WFunTyCon t1) t2 => FunTy (weakTypeToCoreType t1) (weakTypeToCoreType t2)
    | WAppTy  t1 t2                   => match (weakTypeToCoreType t1) with
                                           | TyConApp tc tys => TyConApp tc (app tys ((weakTypeToCoreType t2)::nil))
                                           | t1'             => AppTy t1' (weakTypeToCoreType t2)
                                         end
    | WTyCon    tc                    => TyConApp tc nil
    | WTyFunApp tf lt                 => TyConApp tf (map weakTypeToCoreType lt)
    | WClassP c lt                    => PredTy (ClassP c (map weakTypeToCoreType lt))
    | WIParam n ty                    => PredTy (IParam n (weakTypeToCoreType ty))
    | WForAllTy (weakTypeVar wtv _) t => ForAllTy wtv (weakTypeToCoreType t)
    | WFunTyCon                       => TyConApp ArrowTyCon nil
    | WCodeTy  (weakTypeVar ec _) t   => TyConApp ModalBoxTyCon ((TyVarTy ec)::(weakTypeToCoreType t)::nil)
    | WCoFunTy t1 t2 t3               => FunTy (PredTy (EqPred (weakTypeToCoreType t1) (weakTypeToCoreType t2)))
                                           (weakTypeToCoreType t3)
  end.

Definition weakCoercionToCoreCoercion (wc:WeakCoercion) : CoreCoercion :=
  mkUnsafeCoercion (weakTypeToCoreType (fst (weakCoercionTypes wc))) (weakTypeToCoreType (snd (weakCoercionTypes wc))).

Fixpoint weakExprToCoreExpr (me:WeakExpr) : @CoreExpr CoreVar :=
  match me with
  | WEVar   (weakExprVar v _)           => CoreEVar  v
  | WELit   lit                         => CoreELit  lit
  | WEApp   e1 e2                       => CoreEApp     (weakExprToCoreExpr e1) (weakExprToCoreExpr e2)
  | WETyApp e t                         => CoreEApp     (weakExprToCoreExpr e ) (CoreEType (weakTypeToCoreType t))
  | WECoApp e co                        => CoreEApp     (weakExprToCoreExpr e )
                                                          (CoreEType (coreCoercionsAreReallyTypes (weakCoercionToCoreCoercion co)))
  | WENote  n e                         => CoreENote n  (weakExprToCoreExpr e )
  | WELam   (weakExprVar ev _  ) e      => CoreELam  ev (weakExprToCoreExpr e )
  | WETyLam (weakTypeVar tv _  ) e      => CoreELam  tv (weakExprToCoreExpr e )
  | WECoLam (weakCoerVar cv _ _ _) e    => CoreELam  cv (weakExprToCoreExpr e )
  | WECast  e co                        => CoreECast    (weakExprToCoreExpr e ) (weakCoercionToCoreCoercion co)
  | WEBrak  v (weakTypeVar ec _) e t    => fold_left CoreEApp
                                                 ((CoreEType (TyVarTy ec))::
                                                   (CoreEType (weakTypeToCoreType t))::
                                                   (weakExprToCoreExpr e)::
                                                   nil)
                                                 (CoreEVar v)
  | WEEsc   v (weakTypeVar ec _) e t    => fold_left CoreEApp
```

```
                                          ((CoreEType (TyVarTy ec))::
                                            (CoreEType (weakTypeToCoreType t))::
                                            (weakExprToCoreExpr e)::
                                            nil)
                                          (CoreEVar v)
  | WECSP   v (weakTypeVar ec _) e t     => fold_left CoreEApp
                                          ((CoreEType (TyVarTy ec))::
                                            (CoreEType (weakTypeToCoreType t))::
                                            (weakExprToCoreExpr e)::
                                            nil)
                                          (CoreEVar v)
  | WELet   (weakExprVar v _) ve e       => mkCoreLet      (CoreNonRec v (weakExprToCoreExpr ve))  (weakExprToCoreExpr e)
  | WECase  vscrut escrut tbranches tc types alts  =>
                                      CoreECase (weakExprToCoreExpr escrut) vscrut (weakTypeToCoreType tbranches)
                                        (sortAlts ((
                                          fix mkCaseBranches (alts:Tree
                                            ??(WeakAltCon*list WeakTypeVar*list WeakCoerVar*list WeakExprVar*WeakExpr)) :=
                                          match alts with
                                            | T_Leaf None              => nil
                                            | T_Branch b1 b2           => app (mkCaseBranches b1) (mkCaseBranches b2)
                                            | T_Leaf (Some (ac,tvars,cvars,evars,e)) =>
                                              (mkTriple (weakAltConToCoreAltCon ac)
                                                (app (app
                                                  (map (fun v:WeakTypeVar => weakVarToCoreVar v) tvars)
                                                  (map (fun v:WeakCoerVar => weakVarToCoreVar v) cvars))
                                                (map (fun v:WeakExprVar => weakVarToCoreVar v) evars))
                                                (weakExprToCoreExpr e))::nil
                                          end
                                        ) alts))
  | WELetRec mlr e                       => CoreELet (CoreRec
                                           ((fix mkLetBindings (mlr:Tree ??(WeakExprVar * WeakExpr)) :=
                                            match mlr with
                                              | T_Leaf None                         => nil
                                              | T_Leaf (Some (weakExprVar cv _,e)) => (cv,(weakExprToCoreExpr e))::nil
                                              | T_Branch b1 b2                      => app (mkLetBindings b1) (mkLetBindings b2)
                                            end) mlr))
                                           (weakExprToCoreExpr e)
  end.

Definition weakTypeOfWeakExpr (we:WeakExpr) : ???WeakType :=
  coreTypeToWeakType (coreTypeOfCoreExpr (weakExprToCoreExpr we)).
```

```
Instance weakExprToString : ToString WeakExpr  :=
  { toString := fun we => toString (weakExprToCoreExpr we) }.
```