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

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

Variable tyConOrTyFun : CoreTyCon -> sum TyCon TyFun. Extract Inlined Constant tyConOrTyFun => "tyConOrTyFun".
Variable coreViewDeep : CoreType  -> CoreType.        Extract Inlined Constant coreViewDeep => "coreViewDeep".
Variable coreCoercionToWeakCoercion : CoreCoercion -> WeakCoercion.
  Extract Inlined Constant coreCoercionToWeakCoercion => "coreCoercionToWeakCoercion".

(* extracts the Name from a CoreVar *)
Variable coreVarCoreName    : CoreVar -> CoreName.   Extract Inlined Constant coreVarCoreName  => "Var.varName".

(* the magic wired-in name for the modal type introduction/elimination forms *)
Variable hetmet_brak_name   : CoreName.              Extract Inlined Constant hetmet_brak_name => "PrelNames.hetmet_brak_name".
Variable hetmet_esc_name    : CoreName.              Extract Inlined Constant hetmet_esc_name  => "PrelNames.hetmet_esc_name".
Variable hetmet_csp_name    : CoreName.              Extract Inlined Constant hetmet_csp_name  => "PrelNames.hetmet_csp_name".

Fixpoint coreTypeToWeakType' (ct:CoreType) : ???WeakType :=
  match ct with
  | TyVarTy  cv             => match coreVarToWeakVar cv with
                                 | WExprVar _  => Error "encountered expression variable in a type"
                                 | WTypeVar tv => OK (WTyVarTy tv)
                                 | WCoerVar _  => Error "encountered coercion variable in a type"
                               end

  | AppTy    t1 t2          => coreTypeToWeakType' t2 >>= fun t2' => coreTypeToWeakType' t1 >>= fun t1' => OK (WAppTy t1' t2')

  | TyConApp  tc_ lct       =>
```

```
    let recurse := ((fix rec tl := match tl with
                                   | nil => OK nil
                                   | a::b => coreTypeToWeakType' a >>= fun a' => rec b >>= fun b' => OK (a'::b')
                                 end) lct)
    in match tyConOrTyFun tc_ with
         | inr tf => recurse >>= fun recurse' => OK (WTyFunApp tf recurse')
         | inl tc => if eqd_dec tc ModalBoxTyCon
                       then match lct with
                              | ((TyVarTy ec)::tbody::nil) =>
                                match coreVarToWeakVar ec with
                                  | WTypeVar ec' => coreTypeToWeakType' tbody >>= fun tbody' => OK (WCodeTy ec' tbody')
                                  | WExprVar _  => Error "encountered expression variable in a modal box type"
                                  | WCoerVar _  => Error "encountered coercion variable in a modal box type"
                                end
                              | _                             => Error ("mis-applied modal box tycon: " +++ toString ct)
                            end
                       else let tc' := if eqd_dec tc ArrowTyCon
                                       then WFunTyCon
                                       else WTyCon tc
                            in recurse >>= fun recurse' => OK (fold_left (fun x y => WAppTy x y) recurse' tc')
       end
| FunTy (PredTy (EqPred t1 t2)) t3    => coreTypeToWeakType' t1 >>= fun t1' =>
                                 coreTypeToWeakType' t2 >>= fun t2' =>
                                 coreTypeToWeakType' t3 >>= fun t3' =>
                                   OK (WCoFunTy t1' t2' t3')
| FunTy    t1 t2           => coreTypeToWeakType' t1 >>= fun t1' =>
                              coreTypeToWeakType' t2 >>= fun t2' =>
                                OK (WAppTy (WAppTy WFunTyCon t1') t2')
| ForAllTy cv t             => match coreVarToWeakVar cv with
                                 | WExprVar _  => Error "encountered expression variable in a type"
                                 | WTypeVar tv => coreTypeToWeakType' t >>= fun t' => OK (WForAllTy tv t')
                                 | WCoerVar _  => Error "encountered coercion variable in a type"
                               end
| PredTy (ClassP  cl lct) => ((fix rec tl := match tl with
                                               | nil => OK nil
                                               | a::b => coreTypeToWeakType' a >>= fun a' =>
                                                 rec b >>= fun b' => OK (a'::b')
                                             end) lct) >>= fun lct' => OK (WClassP cl lct')
| PredTy (IParam ipn ct)   => coreTypeToWeakType' ct >>= fun ct' => OK (WIParam ipn ct')
| PredTy (EqPred _ _)   => Error "hit a bare EqPred"
end.
```

```
Fixpoint coreTypeToWeakType t := addErrorMessage "coreTypeToWeakType" (coreTypeToWeakType' (coreViewDeep t)).

(* detects our crude Core-encoding of modal type introduction/elimination forms *)
Definition isBrak (ce:@CoreExpr CoreVar) : ??(WeakExprVar * WeakTypeVar * CoreType) :=
match ce with
  | (CoreEApp (CoreEApp (CoreEVar v) (CoreEType (TyVarTy ec))) (CoreEType tbody))
    => if coreName_eq hetmet_brak_name (coreVarCoreName v) then
      match coreVarToWeakVar ec with
        | WExprVar _  => None
        | WCoerVar _  => None
        | WTypeVar tv => match coreVarToWeakVar v with
                           | WExprVar v' => Some (v',tv,tbody)
                           | _ => None
                         end
      end else None
  | _ => None
end.

Definition isEsc (ce:@CoreExpr CoreVar) : ??(WeakExprVar * WeakTypeVar * CoreType) :=
match ce with
  | (CoreEApp (CoreEApp (CoreEVar v) (CoreEType (TyVarTy ec))) (CoreEType tbody))
    => if coreName_eq hetmet_esc_name (coreVarCoreName v) then
      match coreVarToWeakVar ec with
        | WExprVar _  => None
        | WTypeVar tv => match coreVarToWeakVar v with
                           | WExprVar v' => Some (v',tv,tbody)
                           | _ => None
                         end
        | WCoerVar _  => None
      end else None
  | _ => None
end.

Definition isCSP (ce:@CoreExpr CoreVar) : ??(WeakExprVar * WeakTypeVar * CoreType) :=
match ce with
  | (CoreEApp (CoreEApp (CoreEVar v) (CoreEType (TyVarTy ec))) (CoreEType tbody))
    => if coreName_eq hetmet_csp_name (coreVarCoreName v) then
      match coreVarToWeakVar ec with
        | WExprVar _  => None
        | WTypeVar tv => match coreVarToWeakVar v with
```

```
                          | WExprVar v' => Some (v',tv,tbody)
                          | _ => None
                        end
       | WCoerVar _  => None
      end else None
  | _ => None
end.

(* if this is a (WAppTy (WAppTy ... (WTyCon tc) .. ) .. ), return (tc,[...]) *)
Fixpoint expectTyConApp (wt:WeakType)(acc:list WeakType) : ???(TyCon * list WeakType) :=
  match wt with
    | WTyCon tc        => OK (tc,acc)
    | WAppTy t1 t2     => expectTyConApp t1 (t2::acc)
    | WTyFunApp tc tys => Error ("expectTyConApp encountered TyFunApp: " +++ toString wt)
    | _                => Error ("expectTyConApp encountered " +++ toString wt)
  end.

Fixpoint coreExprToWeakExpr (ce:@CoreExpr CoreVar) : ???WeakExpr :=
  match ce with
    | CoreELit   lit   => OK (WELit lit)
    | CoreENote  n e   => coreExprToWeakExpr e >>= fun e' => OK (WENote n e')
    | CoreEType  t     => Error "encountered CoreEType in a position where an Expr should have been"
    | CoreECast  e co  => coreExprToWeakExpr e >>= fun e' =>
                              OK (WECast e' (coreCoercionToWeakCoercion co))

    | CoreEVar   v     => match coreVarToWeakVar v with
                          | WExprVar ev => OK (WEVar ev)
                          | WTypeVar _  => Error "found a type variable inside an EVar!"
                          | WCoerVar _  => Error "found a coercion variable inside an EVar!"
                        end

    | CoreEApp   e1 e2 => match isBrak e1 with
                          | Some (v,tv,t) =>
                            coreExprToWeakExpr e2 >>= fun e' =>
                              coreTypeToWeakType t >>= fun t' =>
                              OK (WEBrak v tv e' t')
                          | None    => match isEsc e1 with
                                       | Some (v,tv,t) =>
                                         coreExprToWeakExpr e2 >>= fun e' =>
                                           coreTypeToWeakType t >>= fun t' =>
                                             OK (WEEsc v tv e' t')
```

```
                                    | None    => match isCSP e1 with
                                                 | Some (v,tv,t) =>
                                                   coreExprToWeakExpr e2 >>= fun e' =>
                                                     coreTypeToWeakType t >>= fun t' =>
                                                       OK (WECSP v tv e' t')
                                                 | None    => coreExprToWeakExpr e1 >>= fun e1' =>
                                                   match e2 with
                                                     | CoreEType t =>
                                                       coreTypeToWeakType t >>= fun t' =>
                                                         OK (WETyApp e1' t')
                                                     | _           => coreExprToWeakExpr e2
                                                       >>= fun e2' => OK (WEApp e1' e2')
                                                   end
                                               end
                                  end
                      end

| CoreELam   v e => match coreVarToWeakVar v with
                    | WExprVar ev => coreExprToWeakExpr e >>= fun e' => OK (WELam   ev e')
                    | WTypeVar tv => coreExprToWeakExpr e >>= fun e' => OK (WETyLam tv e')
                    | WCoerVar cv => coreExprToWeakExpr e >>= fun e' => OK (WECoLam cv e')
                  end

| CoreELet   (CoreNonRec v ve) e => match coreVarToWeakVar v with
                    | WExprVar ev => coreExprToWeakExpr ve >>= fun ve' =>
                                     coreExprToWeakExpr e  >>= fun e'  => OK (WELet ev ve' e')
                    | WTypeVar _ => match e with
                                    | CoreEType t => Error "saw a type-let"
                                    | _           => Error "saw (ELet <tyvar> e) where e!=EType"
                                    end
                    | WCoerVar _ => Error "saw an (ELet <coercionVar>)"
                  end

| CoreELet   (CoreRec rb)      e =>
  ((fix coreExprToWeakExprList (cel:list (CoreVar * (@CoreExpr CoreVar))) : ???(list (WeakExprVar * WeakExpr)) :=
    match cel with
      | nil => OK nil
      | (v',e')::t => coreExprToWeakExprList t >>= fun t' =>
        match coreVarToWeakVar v' with
          | WExprVar ev => coreExprToWeakExpr e' >>= fun e' => OK ((ev,e')::t')
          | WTypeVar _  => Error "found a type variable in a recursive let"
```

```
          | WCoerVar _  => Error "found a coercion variable in a recursive let"
        end
    end) rb) >>= fun rb' =>
  coreExprToWeakExpr e >>= fun e' =>
  OK (WELetRec (unleaves' rb') e')

| CoreECase  e v tbranches alts =>
  match coreVarToWeakVar v with
    | WTypeVar _  => Error "found a type variable in a case"
    | WCoerVar _  => Error "found a coercion variable in a case"
    | WExprVar ev =>
    coreTypeToWeakType (coreTypeOfCoreExpr e) >>= fun te' =>
      expectTyConApp te' nil >>= fun tca =>
        let (tc,lt) := tca:(TyCon * list WeakType) in
      ((fix mkBranches (branches: list (@triple CoreAltCon (list CoreVar) (@CoreExpr CoreVar)))
            : ???(list (WeakAltCon*list WeakTypeVar*list WeakCoerVar*list WeakExprVar*WeakExpr)) :=
        match branches with
          | nil => OK nil
          | (mkTriple alt vars e)::rest =>
              mkBranches rest >>= fun rest' =>
                coreExprToWeakExpr e >>= fun e' =>
                match alt with
                  | DEFAULT               => OK ((WeakDEFAULT,nil,nil,nil,e')::rest')
                  | LitAlt lit            => OK ((WeakLitAlt lit,nil,nil,nil,e')::rest')
                  | DataAlt dc            => let vars := map coreVarToWeakVar vars in
                    OK (((WeakDataAlt dc),
                    (filter (map (fun x => match x with WTypeVar v => Some v | _ => None end) vars)),
                    (filter (map (fun x => match x with WCoerVar v => Some v | _ => None end) vars)),
                    (filter (map (fun x => match x with WExprVar v => Some v | _ => None end) vars)),
                    e')::rest')
                end
        end) alts)
      >>= fun branches =>
        coreExprToWeakExpr e >>= fun scrutinee =>
          coreTypeToWeakType tbranches >>= fun tbranches' =>
              OK (WECase ev scrutinee tbranches' tc lt (unleaves branches))
  end
end.
```