```
(*****************************************************************************************************************************)
(* HaskCoreTypes: basically GHC's TypeRep.Type imported into Coqland                                                         *)
(*****************************************************************************************************************************)

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

Variable CoreCoercion        : Type.                       Extract Inlined Constant CoreCoercion          => "Coercion.Coercion".
Variable classTyCon          : Class_ -> CoreTyCon.        Extract Inlined Constant classTyCon            => "Class.classTyCon".
Variable coreTyConToString   : CoreTyCon   -> string.      Extract Inlined Constant coreTyConToString     => "outputableToString".
Variable coreDataConToString : CoreDataCon -> string.      Extract Inlined Constant coreDataConToString   => "outputableToString".

(* this exracts onto TypeRep.Type, on the nose *)
Inductive CoreType :=
| TyVarTy  : CoreVar                    -> CoreType
| AppTy    : CoreType  ->      CoreType -> CoreType   (* first arg must be AppTy or TyVarTy*)
| TyConApp : CoreTyCon -> list CoreType -> CoreType
| FunTy    : CoreType  ->      CoreType -> CoreType   (* technically redundant since we have FunTyCon *)
| ForAllTy : CoreVar   ->      CoreType -> CoreType
| PredTy   : PredType                   -> CoreType
with PredType :=
| ClassP   : Class_              -> list CoreType -> PredType
| IParam   : CoreIPName CoreName -> CoreType      -> PredType
| EqPred   : CoreType            -> CoreType      -> PredType.
Extract Inductive CoreType =>
   "TypeRep.Type" [ "TypeRep.TyVarTy" "TypeRep.AppTy" "TypeRep.TyConApp" "TypeRep.FunTy" "TypeRep.ForAllTy" "TypeRep.PredTy" ].
Extract Inductive PredType =>
   "TypeRep.PredType" [ "TypeRep.ClassP" "TypeRep.IParam" "TypeRep.EqPred" ].

Variable coreNameToString      : CoreName     -> string.    Extract Inlined Constant coreNameToString       => "outputableToString".
Variable coreCoercionToString  : CoreCoercion -> string.    Extract Inlined Constant coreCoercionToString   => "outputableToString".
Variable coreCoercionKind : CoreCoercion -> CoreType*CoreType. Extract Inlined Constant coreCoercionKind => "Coercion.coercionKind".
Variable kindOfCoreType   : CoreType -> Kind.   Extract Inlined Constant kindOfCoreType   => "(coreKindToKind . Coercion.typeKind)".
Variable coreTypeToString : CoreType -> string. Extract Inlined Constant coreTypeToString => "(outputableToString . coreViewDeep)".
```

```
(* GHC provides decision procedures for equality on its primitive types; we tell Coq to blindly trust them *)
Variable coreTyCon_eq         : EqDecider CoreTyCon.       Extract Inlined Constant coreTyCon_eq          => "(==)".
Variable tyCon_eq             : EqDecider TyCon.           Extract Inlined Constant tyCon_eq              => "(==)".
Variable tyFun_eq             : EqDecider TyFun.           Extract Inlined Constant tyFun_eq              => "(==)".
Variable dataCon_eq           : EqDecider CoreDataCon.     Extract Inlined Constant dataCon_eq            => "(==)".
Variable coreName_eq          : EqDecider CoreName.        Extract Inlined Constant coreName_eq           => "(==)".
Instance CoreTyConEqDecidable : EqDecidable CoreTyCon   := { eqd_dec := coreTyCon_eq }.
Instance TyConEqDecidable     : EqDecidable TyCon       := { eqd_dec := tyCon_eq }.
Instance TyFunEqDecidable     : EqDecidable TyFun       := { eqd_dec := tyFun_eq }.
Instance DataConEqDecidable   : EqDecidable CoreDataCon := { eqd_dec := dataCon_eq }.
Instance CoreNameEqDecidable  : EqDecidable CoreName    := { eqd_dec := coreName_eq }.
Instance CoreTypeToString     : ToString CoreType       := { toString := coreTypeToString }.
Instance CoreNameToString     : ToString CoreName       := { toString := coreNameToString }.
Instance CoreCoercionToString : ToString CoreCoercion   := { toString := coreCoercionToString }.
Instance CoreDataConToString  : ToString CoreDataCon    := { toString := coreDataConToString }.
Instance CoreTyConToString    : ToString CoreTyCon      := { toString := coreTyConToString }.
```