```
(*****************************************************************************************************************************)
(* HaskLiteralsAndTyCons: representation of compile-time constants (literals)                                                *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Coq.Strings.String.
Require Import HaskKinds.

Variable CoreDataCon     : Type.                    Extract Inlined Constant CoreDataCon          => "DataCon.DataCon".

(* once again, we pull the trick of having multiple Coq types map to a single Haskell type to provide stronger typing *)
Variable TyCon           : Type.                    Extract Inlined Constant TyCon                => "TyCon.TyCon".
Variable TyFun           : Type.                    Extract Inlined Constant TyFun                => "TyCon.TyCon".

(* Since GHC is written in Haskell, compile-time Haskell constants are represented using Haskell (Prelude) types *)
Variable HaskInt         : Type.                    Extract Inlined Constant HaskInt               => "Prelude.Int".
Variable HaskChar        : Type.                    Extract Inlined Constant HaskChar              => "Prelude.Char".
Variable HaskFastString  : Type.                    Extract Inlined Constant HaskFastString        => "FastString.FastString".
Variable HaskInteger     : Type.                    Extract Inlined Constant HaskInteger           => "Prelude.Integer".
Variable HaskRational    : Type.                    Extract Inlined Constant HaskRational          => "Prelude.Rational".

Variable CoreName        : Type.                    Extract Inlined Constant CoreName              => "Name.Name".
Variable Class_          : Type.                    Extract Inlined Constant Class_                => "Class.Class".
Variable CoreIPName      : Type -> Type.            Extract         Constant CoreIPName "'a"       => "BasicTypes.IPName".
                                                    Extraction Inline CoreIPName.

(* This type extracts exactly onto GHC's Literal.Literal type *)
Inductive HaskLiteral :=
| HaskMachChar     : HaskChar                                                  -> HaskLiteral
| HaskMachStr      : HaskFastString                                            -> HaskLiteral
| HaskMachNullAddr :                                                             HaskLiteral
| HaskMachInt      : HaskInteger                                               -> HaskLiteral
| HaskMachInt64    : HaskInteger                                               -> HaskLiteral
| HaskMachWord     : HaskInteger                                               -> HaskLiteral
| HaskMachWord64   : HaskInteger                                               -> HaskLiteral
| HaskMachFloat    : HaskRational                                              -> HaskLiteral
| HaskMachDouble   : HaskRational                                              -> HaskLiteral
| HaskMachLabel    : HaskFastString -> option HaskInt -> HaskFunctionOrData -> HaskLiteral
with HaskFunctionOrData : Type := HaskIsFunction | HaskIsData.
```

```
Extract Inductive HaskLiteral => "Literal.Literal"
  [ "Literal.MachChar"
    "Literal.MachStr"
    "Literal.MachNullAddr"
    "Literal.MachInt"
    "Literal.MachInt64"
    "Literal.MachWord"
    "Literal.MachWord64"
    "Literal.MachFloat"
    "Literal.MachDouble"
    "Literal.MachLabel" ].
Extract Inductive HaskFunctionOrData =>
  "BasicTypes.FunctionOrData" [ "BasicTypes.IsFunction" "BasicTypes.IsData" ].

Variable haskLiteralToString : HaskLiteral -> string.    Extract Inlined Constant haskLiteralToString     => "outputableToString".
Instance HaskLiteralToString : ToString HaskLiteral := { toString := haskLiteralToString }.

(* the TyCons for each of the literals above *)
Variable addrPrimTyCon       : TyCon.   Extract Inlined Constant addrPrimTyCon   => "TysPrim.addrPrimTyCon".
Variable intPrimTyCon        : TyCon.   Extract Inlined Constant intPrimTyCon    => "TysPrim.intPrimTyCon".
Variable wordPrimTyCon       : TyCon.   Extract Inlined Constant wordPrimTyCon   => "TysPrim.wordPrimTyCon".
Variable int64PrimTyCon      : TyCon.   Extract Inlined Constant int64PrimTyCon  => "TysPrim.int64PrimTyCon".
Variable word64PrimTyCon     : TyCon.   Extract Inlined Constant word64PrimTyCon => "TysPrim.word64PrimTyCon".
Variable floatPrimTyCon      : TyCon.   Extract Inlined Constant floatPrimTyCon  => "TysPrim.floatPrimTyCon".
Variable doublePrimTyCon     : TyCon.   Extract Inlined Constant doublePrimTyCon => "TysPrim.doublePrimTyCon".
Variable charPrimTyCon       : TyCon.   Extract Inlined Constant charPrimTyCon   => "TysPrim.charPrimTyCon".

(* retrieves the TyCon for a given Literal *)
Definition haskLiteralToTyCon (lit:HaskLiteral) : TyCon :=
match lit with
  | HaskMachNullAddr    => addrPrimTyCon
  | HaskMachChar _      => charPrimTyCon
  | HaskMachStr  _      => addrPrimTyCon
  | HaskMachInt  _      => intPrimTyCon
  | HaskMachWord  _     => wordPrimTyCon
  | HaskMachInt64  _    => int64PrimTyCon
  | HaskMachWord64  _   => word64PrimTyCon
  | HaskMachFloat _     => floatPrimTyCon
  | HaskMachDouble _    => doublePrimTyCon
  | HaskMachLabel _ _ _ => addrPrimTyCon
```

```
end.

Variable tyConToString   : TyCon   -> string.     Extract Inlined Constant tyConToString         => "outputableToString".
Variable tyFunToString   : TyFun   -> string.     Extract Inlined Constant tyFunToString         => "outputableToString".
Instance TyConToString   : ToString TyCon := { toString := tyConToString }.
Instance TyFunToString   : ToString TyFun := { toString := tyFunToString }.
Instance TyConToLatex    : ToLatex  TyCon := { toLatex  := fun x => toLatex (toString x) }.
Instance TyFunToLatex    : ToLatex  TyCon := { toLatex  := fun x => toLatex (toString x) }.

Variable ModalBoxTyCon   : TyCon.        Extract Inlined Constant ModalBoxTyCon => "TysWiredIn.hetMetCodeTypeTyCon".
Variable ArrowTyCon      : TyCon.        Extract Constant ArrowTyCon    => "Type.funTyCon".
```