```
(*****************************************************************************************************************************)
(* HaskKinds:  Definitions shared by all four representations (Core, Weak, Strong, Proof)                                    *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import Coq.Strings.String.
Open Scope nat_scope.

Variable Note                : Type.                          Extract Inlined Constant Note        => "CoreSyn.Note".
Variable natToString         : nat         -> string.         Extract Inlined Constant natToString => "natToString".
Instance NatToStringInstance : ToString nat := { toString := natToString }.

Inductive Kind : Type :=
  | KindStar          : Kind                      (* ★ - the kind of coercions and the kind of types inhabited by [boxed] values *)
  | KindArrow         : Kind  -> Kind  -> Kind    (* ⇛ - type-function-space; things of kind X⇛Y are NOT inhabited by expressions*)
  | KindUnliftedType  : Kind                      (* not in the paper - this is the kind of unboxed non-tuple types *)
  | KindUnboxedTuple  : Kind                      (* not in the paper - this is the kind of unboxed tuples *)
  | KindArgType       : Kind                      (* not in the paper - this is the lub of KindStar and KindUnliftedType *)
  | KindOpenType      : Kind                      (* not in the paper - kind of all types (lifted, boxed, whatever) *)
  .

Open Scope string_scope.
Fixpoint kindToString (k:Kind) : string :=
  match k with
  | KindStar                    => "*"
  | KindArrow KindStar k2       => "*=>"+++kindToString k2
  | KindArrow k1 k2             => "("+++kindToString k1+++")=>"+++kindToString k2
  | KindUnliftedType            => "#"
  | KindUnboxedTuple            => "(#)"
  | KindArgType                 => "??"
  | KindOpenType                => "?"
  end.
Instance KindToString : ToString Kind := { toString := kindToString }.

Notation "'★"   := KindStar.
Notation "a ⇛b" := (KindArrow a b).

Fixpoint kindToLatexMath (k:Kind) : LatexMath :=
  match k with
```

```
  | ★                          => rawLatexMath "\star"
  | ★ ⇛k2                      => (rawLatexMath "\star\Rightarrow ")+++kindToLatexMath k2
  | k1 ⇛k2                     => (rawLatexMath "(")+++kindToLatexMath k1+++(rawLatexMath ")\Rightarrow ")+++kindToLatexMath k2
  | KindUnliftedType           => rawLatexMath "\text{\tt{\#}}"
  | KindUnboxedTuple           => rawLatexMath "\text{\tt{(\#)}}"
  | KindArgType                => rawLatexMath "\text{\tt{??}}"
  | KindOpenType               => rawLatexMath "\text{\tt{?}}"
  end.
Instance KindToLatexMath : ToLatexMath Kind := { toLatexMath := kindToLatexMath }.

Instance KindEqDecidable : EqDecidable Kind.
  apply Build_EqDecidable.
  induction v1.
    destruct v2; try (right; intro q; inversion q; fail) ; left ; auto.
    destruct v2; try (right; intro q; inversion q; fail) ; auto.
      destruct (IHv1_1 v2_1); destruct (IHv1_2 v2_2); subst; auto;
      right; intro; subst; inversion H; subst; apply n; auto.
    destruct v2; try (right; intro q; inversion q; fail) ; left ; auto.
    destruct v2; try (right; intro q; inversion q; fail) ; left ; auto.
    destruct v2; try (right; intro q; inversion q; fail) ; left ; auto.
    destruct v2; try (right; intro q; inversion q; fail) ; left ; auto.
  Defined.

(* splits a kind into a list of arguments and a result *)
Fixpoint splitKind (k:Kind) : ((list Kind) * Kind) :=
  match k with
    | a ⇛b => let (args,res) := splitKind b in ((a::args),res)
    | _     => (nil, k)
  end.
```