```
(*****************************************************************************************************************************)
(* HaskProofToLatex: render HaskProof's as LaTeX code using trfrac.sty                                                       *)
(*****************************************************************************************************************************)

Generalizable All Variables.
Require Import Preamble.
Require Import General.
Require Import NaturalDeduction.
Require Import Coq.Strings.String.
Require Import Coq.Lists.List.
Require Import HaskKinds.
Require Import HaskWeakVars.
Require Import HaskWeakTypes.
Require Import HaskLiteralsAndTyCons.
Require Import HaskStrongTypes.
Require Import HaskStrong.
Require Import HaskProof.
Require Import HaskCoreTypes.



Inductive VarNameStore :=
  varNameStore : nat -> nat -> nat -> VarNameStore.
Inductive VarNameStoreM {T} :=
  varNameStoreM : (VarNameStore -> (T * VarNameStore)) -> VarNameStoreM.
Implicit Arguments VarNameStoreM [ ].

Open Scope nat_scope.
Instance VarNameMonad : Monad VarNameStoreM :=
{ returnM := fun _   x   => varNameStoreM (fun v => (x,v))
; bindM   := fun _ _ x f =>
  match x with
    varNameStoreM xf =>
  varNameStoreM (fun vns =>
    match xf vns with
      (x',vns') => match f x' with
                     varNameStoreM fx' => fx' vns'
                   end
    end) end }.

Definition freshTyVarName : Kind -> VarNameStoreM LatexMath :=
```

```
  fun κ => varNameStoreM (fun vns =>
    match vns with
      varNameStore n1 n2 n3 =>
      let name := (rawLatexMath "\alpha_{")+++toLatexMath (toString n1)+++(rawLatexMath "}")
        in (name,(varNameStore (S n1) n2 n3))
    end).
Definition freshCoVarName : VarNameStoreM LatexMath :=
  varNameStoreM (fun vns =>
    match vns with
      varNameStore n1 n2 n3 =>
      let name := (rawLatexMath "\gamma_{")+++toLatexMath (toString n2)+++(rawLatexMath "}")
        in (name,(varNameStore n1 (S n2) n3))
    end).
Definition freshValVarName : VarNameStoreM LatexMath :=
  varNameStoreM (fun vns =>
    match vns with
      varNameStore n1 n2 n3 =>
      let vn := match n3 with
                  | 0 => "x"
                  | 1 => "y"
                  | 2 => "z"
                  | 3 => "a"
                  | 4 => "b"
                  | 5 => "c"
                  | 6 => "d"
                  | 7 => "e"
                  | 8 => "f"
                  | 9 => "g"
                  | 10 => "h"
                  | S (S (S (S (S (S (S (S (S (S x)))))))))) => "z_{"+++toString x+++"}"
                end
      in let name := rawLatexMath ("x_{"+++vn+++"}")
        in (name,(varNameStore n1 n2 (S n3)))
    end).

Fixpoint typeToLatexMath (needparens:bool){κ}(t:RawHaskType (fun _ => LatexMath) κ) {struct t} : VarNameStoreM LatexMath :=
  match t return VarNameStoreM LatexMath with
  | TVar    _ v        => return toLatexMath v
  | TCon    tc         => return ((rawLatexMath "\text{\tt{")+++toLatexMath (toString tc)+++(rawLatexMath "}}"))
  | TArrow             => return rawLatexMath "\text{\tt{(->)}}"
  | TCoerc _ t1 t2   t => bind t1' = typeToLatexMath false      t1
```

```
                      ; bind t2' = typeToLatexMath false      t2
                      ; bind t'  = typeToLatexMath needparens t
                      ; return ((rawLatexMath "{(")+++t1'+++(rawLatexMath "{\sim}")+++
                                t2'+++(rawLatexMath ")}\Rightarrow{")+++t'+++(rawLatexMath "}"))
  | TCode  ec t       => bind ec' = typeToLatexMath true ec
                      ; bind t'  = typeToLatexMath false t
                      ; return (rawLatexMath "\code{")+++ec'+++(rawLatexMath "}{")+++t'+++(rawLatexMath "}")
  | TAll   k f        => bind alpha = freshTyVarName k
                      ; bind t'    = typeToLatexMath false (f alpha)
                      ; return (rawLatexMath "(\forall ")+++alpha+++(rawLatexMath "{:}")+++
                      (kindToLatexMath k)+++(rawLatexMath ")")+++t'
  | TApp  _ _  t1 t2  => match t1 return VarNameStoreM LatexMath with
                | TApp _ _ TArrow tx => bind t1' = typeToLatexMath true tx
                                      ; bind t2' = typeToLatexMath true t2
                                      ; let body := t1'+++(rawLatexMath "{\rightarrow}")+++t2'
                                        in return (if needparens then (rawLatexMath "(")+++body+++(rawLatexMath ")") else body)
                | _                 => bind t1' = typeToLatexMath true t1
                                      ; bind t2' = typeToLatexMath true t2
                                      ; let body := t1'+++(rawLatexMath " ")+++t2'
                                        in return (if needparens then (rawLatexMath "(")+++body+++(rawLatexMath ")") else body)
                end
  | TyFunApp   tfc lt => bind rest = typeListToRawLatexMath false lt
                      ; return (rawLatexMath "{\text{\tt{")+++(toLatexMath (toString tfc))+++(rawLatexMath "}}}")+++
                               (rawLatexMath "_{")+++(rawLatexMath (toString (length (fst (tyFunKind tfc)))))+++
                               (rawLatexMath "}")+++
                               (fold_left (fun x y => (rawLatexMath " \  ")+++x+++y)
                                 rest (rawLatexMath ""))
end
with typeListToRawLatexMath (needparens:bool){κ}(t:RawHaskTypeList κ) {struct t} : VarNameStoreM (list LatexMath) :=
match t return VarNameStoreM (list LatexMath) with
| TyFunApp_nil                 => return nil
| TyFunApp_cons  κ kl rhk rhkl => bind r  = typeToLatexMath needparens rhk
                                ; bind rl = typeListToRawLatexMath needparens rhkl
                                ; return (r::rl)
end.

Definition ltypeToLatexMath {Γ:TypeEnv}{κ}(t:LeveledHaskType Γ κ) : VarNameStoreM LatexMath.
  refine (
  match t with
    t' @@ lev =>
    bind ite = _ ;
```

```
    bind t'' = typeToLatexMath false (t' (fun _ => LatexMath) ite) ;
    return match lev with
    | nil => t''
    | lv  => (rawLatexMath " ")+++t''+++(rawLatexMath " @ ")+++
          (fold_left (fun x y => x+++(rawLatexMath ":")+++y)
            (map (fun l:HaskTyVar Γ ★=> l (fun _ => LatexMath) ite) lv) (rawLatexMath ""))
    end
    end); try apply ConcatenableLatexMath.
    try apply VarNameMonad.
    clear t t' lev κ.
    induction Γ.
    apply (return INil).
    refine (bind tv' = freshTyVarName a ; _).
    apply VarNameMonad.
    refine (bind IHΓ' = IHΓ ; return _).
    apply VarNameMonad.
    apply ICons.
    apply tv'.
    apply IHΓ'.
    apply VarNameMonad.
    Defined.

Definition judgmentToRawLatexMath (j:Judg) : LatexMath :=
  match match j return VarNameStoreM LatexMath with
    | mkJudg Γ Δ Σ₁ Σ₂ =>
         bind Σ₁' = treeM (mapOptionTree ltypeToLatexMath Σ₁)
       ; bind Σ₂' = treeM (mapOptionTree ltypeToLatexMath Σ₂)
       ; return treeToLatexMath Σ₁' +++ (rawLatexMath "\vdash") +++ treeToLatexMath Σ₂'
  end with
    varNameStoreM f => fst (f (varNameStore 0 0 0))
  end.

Instance ToLatexMathJudgment : ToLatexMath Judg :=
  { toLatexMath := judgmentToRawLatexMath }.

Fixpoint nd_uruleToRawLatexMath {T}{h}{c}(r:@Arrange T h c) : string :=
  match r with
    | RLeft   _ _ _ r => nd_uruleToRawLatexMath r
    | RRight  _ _ _ r => nd_uruleToRawLatexMath r
    | RCanL   _     => "CanL"
    | RCanR   _     => "CanR"
```

```
    | RuCanL  _     => "uCanL"
    | RuCanR  _     => "uCanR"
    | RAssoc  _ _ _ => "Assoc"
    | RCossa  _ _ _ => "Cossa"
    | RExch   _ _   => "Exch"
    | RWeak   _     => "Weak"
    | RCont   _     => "Cont"
    | RComp   _ _ _ _ _  => "Comp"  (* FIXME: do a better job here *)
  end.

Fixpoint nd_ruleToRawLatexMath {h}{c}(r:Rule h c) : string :=
  match r with
    | RArrange      _ _ _ _ _ r       => nd_uruleToRawLatexMath r
    | RNote         _ _ _ _ _ _       => "Note"
    | RLit          _ _ _ _           => "Lit"
    | RVar          _ _ _ _           => "Var"
    | RGlobal       _ _ _ _ _         => "Global"
    | RLam          _ _ _ _ _ _       => "Abs"
    | RCast         _ _ _ _ _ _ _     => "Cast"
    | RAbsT         _ _ _ _ _ _       => "AbsT"
    | RAppT         _ _ _ _ _ _ _     => "AppT"
    | RAppCo        _ _ _ _ _ _ _ _ _ => "AppCo"
    | RAbsCo        _ _ _ _ _ _ _ _   => "AbsCo"
    | RApp          _ _ _ _ _ _ _     => "App"
    | RLet          _ _ _ _ _ _ _     => "Let"
    | RJoin _ _ _ _ _ _       => "RJoin"
    | RLetRec       _ _ _ _ _ _       => "LetRec"
    | RCase         _ _ _ _ _ _ _ _   => "Case"
    | RBrak         _ _ _ _ _ _       => "Brak"
    | REsc          _ _ _ _ _ _       => "Esc"
    | RVoid   _ _               => "RVoid"
end.

Fixpoint nd_hideURule {T}{h}{c}(r:@Arrange T h c) : bool :=
  match r with
    | RLeft   _ _ _ r            => nd_hideURule r
    | RRight  _ _ _ r            => nd_hideURule r
    | RCanL   _                  => true
    | RCanR   _                  => true
    | RuCanL  _                  => true
    | RuCanR  _                  => true
```

```
    | RAssoc  _ _ _               => true
    | RCossa  _ _ _               => true
    | RExch      (T_Leaf None) b  => true
    | RExch   a  (T_Leaf None)    => true
    | RWeak      (T_Leaf None)    => true
    | RCont      (T_Leaf None)    => true
    | RComp   _ _ _ _ _           => false   (* FIXME: do better *)
    | _                           => false
  end.
Fixpoint nd_hideRule {h}{c}(r:Rule h c) : bool :=
  match r with
    | RArrange      _ _ _ _ _ r => nd_hideURule r
    | RVoid   _ _         => true
    | RJoin _ _ _ _ _ _ => true
    | _                         => false
  end.

Instance ToLatexMathRule h c : ToLatexMath (Rule h c) :=
  { toLatexMath := fun r => rawLatexMath (@nd_ruleToRawLatexMath h c r) }.

Definition nd_ml_toLatexMath {c}(pf:@ND _ Rule [] c) :=
@SIND_toLatexMath _ _ ToLatexMathJudgment ToLatexMathRule (@nd_hideRule) _ _
  (mkSIND (systemfc_all_rules_one_conclusion) _ _ _ pf (scnd_weak [])).
```