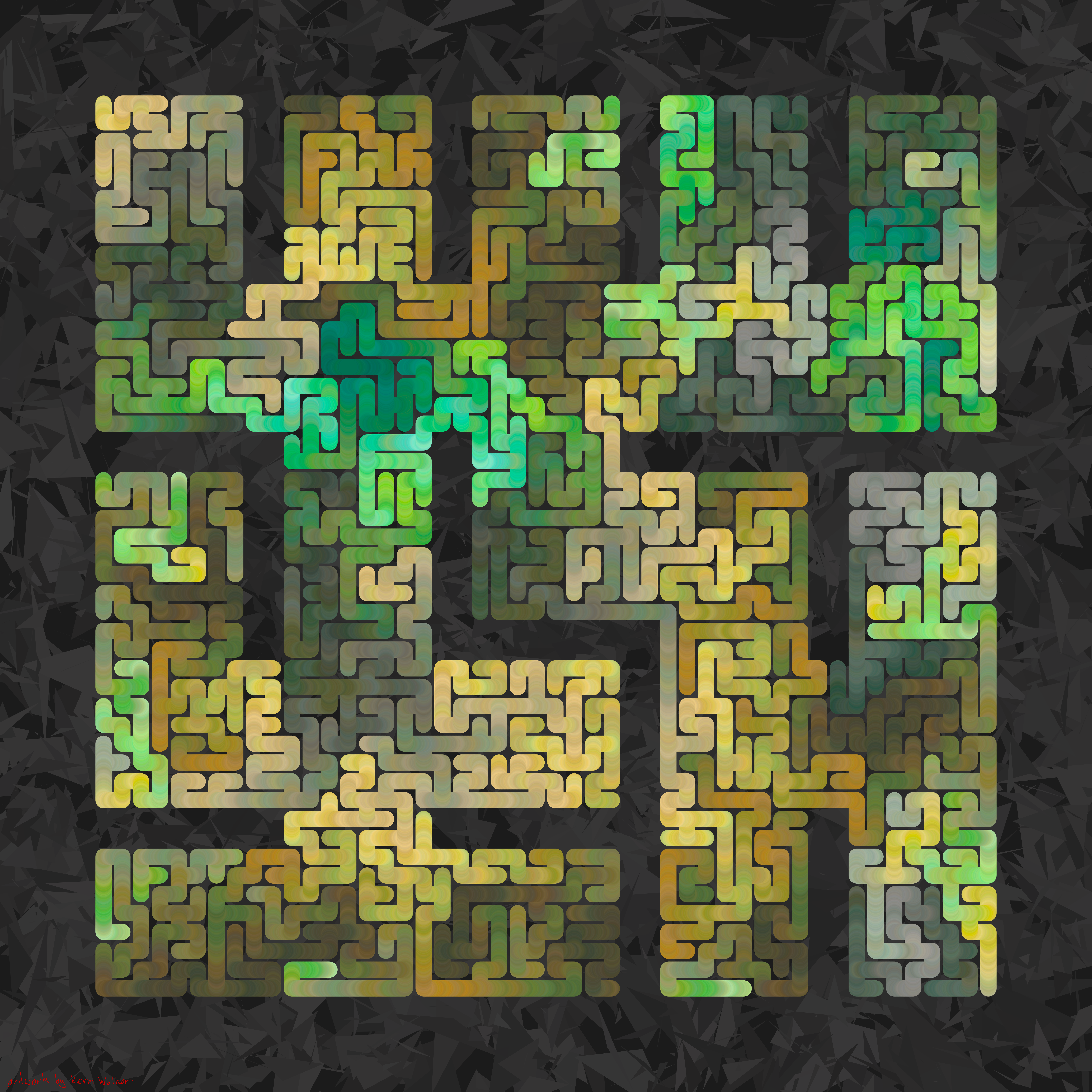
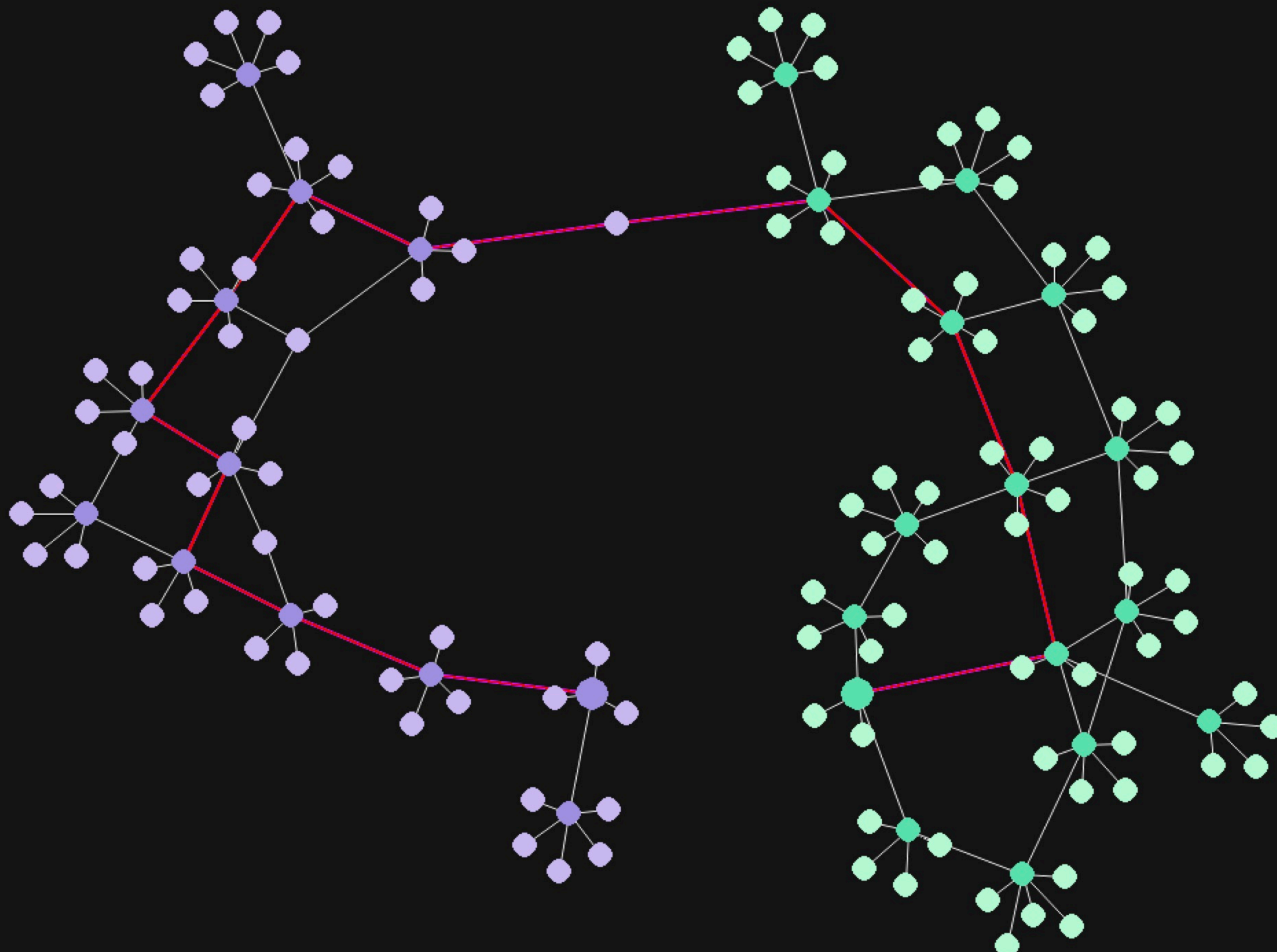

artwork by Kevin Walker



interactive theorem proving

why?!

what is it?

how do I do it?

where next?

interactive theorem proving

why?!

- computers can help us create and understand mathematics
- formalisation reduces the burden of verification, and increases confidence in correctness
- the constructive fragment of formalisation bridges the gap between proofs and calculations
- perhaps (hopefully!) its an integral part of our future
- students love it!

interactive theorem proving

what is it?

- a family of high level programming languages
- expressive enough to encompass modern mathematics
- tooling and user interface to manage goals, hypotheses, axioms, theorems, scopes, dependencies ....
- 'automation': programs that write proofs

interactive theorem proving

what is it?

Examples: Mizar, Isabelle, Coq.

Today: Lean

- based on dependent type theory
- it's the latest and greatest — and changing underneath you!
- open source, developed at Microsoft Research, active community
- Lean is its own metalangauge

A crash course in dependent type theory.

- everything is a term: 3, ["a", " ", "l", "i", "s", "t"], S^2, $\mathbb{N}$
  or a type: $\mathbb{N}$, list string, smooth_manifold, Type 1
- every term has an unambiguous and fixed type.
- there is an effective procedure for typechecking.
- "propositions as types":

  is_prime 57 is a type

  and a term of that type would be a proof.

  $\Longrightarrow$ writing a proof is the same thing as constructing a function.

A crash course in dependent type theory.

- we can construct new types:

① def vector $(\alpha : \text{Type})\ (n : \mathbb{N}) : \text{Type} := \{ L : \text{list } \alpha \mid L.\text{length} = n \}$

② inductive labelled_tree $(\beta : \text{Type}) : \text{Type}$
| leaf : $\beta \longrightarrow$ labelled_tree
| branch : $\beta \longrightarrow$ labelled_tree $\longrightarrow$ labelled_tree $\longrightarrow$ labelled_tree

③ structure Presheaf $(C : \text{Type})$ [category C] :=
$(X : \text{Top})$
$(\mathcal{O} : \text{open\_sets } X \Rightarrow C)$

A crash course in <u>dependent type theory</u>.

- very similar to the logical foundations of Coq ("calculus of inductive constructions")
- Lean has a model in ZFC + inaccessible cardinals
- I think dependent type theory comes very naturally to mathematicians (possibly more so than ZFC: "3 is a topology on 2")
- No commitment to constructivity, intuitionistic logic, or homotopy type theory, although they're available.

interactive theorem proving

- Lean can run in a browser
- Runs in cocalc
  - collaborative editor
  - course management
- Run locally with editor support in VS Code or emacs.

how do I do it?

- There's an introductory book for mathematicians "Theorem proving in Lean".

Live demo — there are infinitely many primes.

```
theorem infinitude_of_primes (N : ℕ) : ∃ p ≥ N, prime p :=
begin
  let M := fact N + 1,
  let p := min_fac M,
  have pp : prime p :=
    min_fac_prime (ne_of_gt (succ_lt_succ (fact_pos N))),
  existsi p,
  split,
  {
    by_contradiction,
    simp at a,
    have h₁ : p ∣ M, apply min_fac_dvd,
    have h₂ : p ∣ fact N :=
      dvd_fact (prime.pos pp) (le_of_lt a),
    have h : p ∣ 1 := dvd_add_right h₂ h₁,
    exact prime.not_dvd_one pp h,
  },
  exact pp
end
```

interactive theorem proving

where next?

where next?

- **mathlib**, the standard library for Lean is primitive but growing fast (Examples: holomorphic functions, Noetherian rings, and the Yoneda lemma all in the last month.)

- We're increasingly confident it's possible to do modern mathematics — perfectoid spaces are 'almost ready'

- students are getting involved —
  - undergrad research projects at ANU and Imperial
  - homework sets in Lean at Imperial this semester!

where next?

- Formalisation is creeping towards relevance

A CORRECTED QUANTITATIVE VERSION OF THE MORSE LEMMA

SÉBASTIEN GOUËZEL AND VLADIMIR SHCHUR

ABSTRACT. There is a gap in the proof of the main theorem in the article [Shc13a] on optimal bounds for the Morse lemma in Gromov-hyperbolic spaces. We correct this gap, showing that the main theorem of [Shc13a] is correct. We also describe a computer certification of this result.

The new proof of Theorem 1.1 has been completely formalized in Isabelle/HOL in [Gou18]. Therefore, the above theorem is certified. Here is this statement as proved in Isabelle/HOL.

```
theorem (in Gromov_hyperbolic_space) Morse_Gromov_theorem':
  fixes f::"real ⇒ 'a"
  assumes "lambda C-quasi_isometry_on {a..b} f"
          "geodesic_segment_between G (f a) (f b)"
  shows "hausdorff_distance (f`{a..b}) G ≤ 92 * lambda^2 * (C + deltaG(TYPE('a)))"
```

- A major "formal abstracts" project will next year start formalising abstracts of major papers in Lean.

where next?

- Lean makes it significantly easier to write new tactics ('programs that write proofs').
- It's still way too cumbersome to write mathematics in Lean.
- As it becomes possible for mathematicians to write tactics (not just langauge developers), this may rapidly change!

where next?

Category theory in Lean

- Can we write enough automation, so that we can write 'human-like' proofs?

(i.e. omitting lots of detail!)

Unimath:

```
(** * Yoneda functor *)

(** ** On objects *)

Definition yoneda_objects_ob (C : precategory) (c : C)
          (d : C) := hom C d c.

Definition yoneda_objects_mor (C : precategory) (c : C)
    (d d' : C) (f : hom C d  d') :
   yoneda_objects_ob C c d' -> yoneda_objects_ob C c d :=
    λ g, f · g.

Definition yoneda_ob_functor_data (C : precategory) (hs: has_homsets C) (c : C) :
    functor_data (C^op) HSET.
Proof.
  exists (λ c', hSetpair (yoneda_objects_ob C c c') (hs c' c)) .
  intros a b f g. unfold yoneda_objects_ob in *. simpl in *.
  exact (f · g).
Defined.


Lemma is_functor_yoneda_functor_data (C : precategory) (hs: has_homsets C) (c : C) :
  is_functor (yoneda_ob_functor_data C hs c).
Proof.
  repeat split; unf; simpl.
  unfold functor_idax .
  intros.
  apply funextsec.
  intro f. unf. apply id_left.
  intros a b d f g.
  apply funextsec. intro h.
  apply (! assoc _ _ _ ).
Qed.

Definition yoneda_objects (C : precategory) (hs: has_homsets C) (c : C) :
             functor C^op HSET :=
    tpair _ _ (is_functor_yoneda_functor_data C hs c).


(** ** On morphisms *)

Definition yoneda_morphisms_data (C : precategory)(hs: has_homsets C) (c c' : C)
    (f : hom C c c') : ∏ a : ob C^op,
         hom   (voneda obiects C hs c a) ( voneda obiects C hs c' a) :=
```

Coq:

```
Section yoneda_lemma.
  Context `{Funext}.
  Variable A : PreCategory.
  Variable G : object (A^op -> set_cat).
  Variable a : A.
    (** There is a contravariant version of Yoneda's lemma which
        concerns contravariant functors from [A] to [Set]. This
        version involves the contravariant hom-functor

        [hₐ = Hom(—, A)],

        which sends [x] to the hom-set [Hom(x, a)]. Given an arbitrary
        contravariant functor [G] from [A] to [Set], Yoneda's lemma
        asserts that

        [Nat(hₐ, G) ≅ G(a)]. *)

  Definition yoneda_lemma_morphism
  : morphism set_cat
             (BuildhSet
                (morphism (A^op -> set_cat) (yoneda A a) G)
                _)
             (G a)
    := fun phi => phi a 1%morphism.

  Local Arguments Overture.compose / .

  Definition yoneda_lemma_morphism_inverse
  : morphism set_cat
             (G a)
             (BuildhSet
                (morphism (A^op -> set_cat) (yoneda A a) G)
                _).
  Proof.
    intro Ga.
    hnf.
    let F0 := match goal with |- NaturalTransformation ?F ?G => constr:(F) end in
    let G0 := match goal with |- NaturalTransformation ?F ?G => constr:(G) end in
    refine (Build_NaturalTransformation
              F0 G0
              (fun a' : A => (fun f : morphism A a' a => morphism_of G f Ga))
              _
           ).
    simpl in *.
    abstract (
```

Isabelle:

```
theory Yoneda
imports NatTrans SetCat
begin

definition YFtorNT' C f ≡ (|NTDom = Hom_C[−,dom_C f] , NTCod = Hom_C[−,cod_C
f] ,
                NatTransMap = λ B . Hom_C[B,f]|)

definition YFtorNT C f ≡ MakeNT (YFtorNT' C f)

lemmas YFtorNT-defs = YFtorNT'-def YFtorNT-def MakeNT-def

lemma YFtorNTCatDom: NTCatDom (YFtorNT C f) = Op C
by (simp add: YFtorNT-defs NTCatDom-def HomFtorContraDom)

lemma YFtorNTCatCod: NTCatCod (YFtorNT C f) = SET
by (simp add: YFtorNT-defs NTCatCod-def HomFtorContraCod)

lemma YFtorNTApp1: assumes X ∈ Obj (NTCatDom (YFtorNT C f)) shows
(YFtorNT C f) $$ X = Hom_C[X,f]
proof−
  have (YFtorNT C f) $$ X = (YFtorNT' C f) $$ X using assms by (simp add:
MakeNTApp YFtorNT-def)
  thus ?thesis by (simp add: YFtorNT'-def)
qed

definition
  YFtor' C ≡ (|
        CatDom = C ,
        CatCod = CatExp (Op C) SET ,
        MapM = λ f . YFtorNT C f
  |)

definition YFtor C ≡ MakeFtor(YFtor' C)

lemmas YFtor-defs = YFtor'-def YFtor-def MakeFtor-def

lemma YFtorNTNatTrans':
  assumes LSCategory C and f ∈ Mor C
  shows NatTransP (YFtorNT' C f)
proof(auto simp only: NatTransP-def)
  have Fd: Ftor (NTDom (YFtorNT' C f)) : (Op C) ⟶ SET using assms
    by (simp add:  HomFtorContraFtor Category.Cdom YFtorNT'-def)
  have Fc: Ftor (NTCod (YFtorNT' C f)) : (Op C) ⟶ SET using assms
    by (simp add:  HomFtorContraFtor Category.Ccod YFtorNT'-def)
  show Functor (NTDom (YFtorNT' C f)) using Fd by auto
  show Functor (NTCod (YFtorNT' C f)) using Fc by auto
  show NTCatDom (YFtorNT' C f) = CatDom (NTCod (YFtorNT' C f))
    by(simp add: YFtorNT'-def NTCatDom-def HomFtorContraDom)
  show NTCatCod (YFtorNT' C f) = CatCod (NTDom (YFtorNT' C f))

        MapM = λ f . YFtorNT C f
  |)
```

```
variables (C : Type u₁) [𝒞 : category.{u₁ v₁} C]
include 𝒞

def yoneda : C ⥤ ((Cᵒᵖ) ⥤ (Type v₁)) := λ X, λ Y : C, Y ⟶ X.

def yoneda_evaluation : (((Cᵒᵖ) ⥤ (Type v₁)) × (Cᵒᵖ)) ⥤ (Type (max u₁ v₁)) :=
(evaluation (Cᵒᵖ) (Type v₁)) ⋙ ulift_functor.{v₁ u₁}

@[simp] lemma yoneda_evaluation_map_down
  (P Q : (Cᵒᵖ ⥤ Type v₁) ×  (Cᵒᵖ)) (α : P ⟶ Q) (x : (yoneda_evaluation C) P) :
  ((yoneda_evaluation C).map α x).down = (α.1) (Q.2) ((P.1).map (α.2) (x.down)) := rfl

def yoneda_pairing : (((Cᵒᵖ) ⥤ (Type v₁)) × (Cᵒᵖ)) ⥤ (Type (max u₁ v₁)) :=
let F := (category_theory.prod.swap ((Cᵒᵖ) ⥤ (Type v₁)) (Cᵒᵖ)) in
let G := (functor.prod ((yoneda C).op) (functor.id ((Cᵒᵖ) ⥤ (Type v₁)))) in
let H := (functor.hom ((Cᵒᵖ) ⥤ (Type v₁))) in
  (F ⋙ G ⋙ H)

@[simp] lemma yoneda_pairing_map
  (P Q : (Cᵒᵖ ⥤ Type v₁) ×  (Cᵒᵖ)) (α : P ⟶ Q) (β : (yoneda_pairing C) (P.1, P.2)) :
  (yoneda_pairing C).map α β = (yoneda C).map (α.snd) ≫ β ≫ α.fst := rfl

def yoneda_lemma : (yoneda_pairing C) ≅ (yoneda_evaluation C) :=
{ hom := { app := λ F x, ulift.up ((x.app F.2) (𝟙 F.2)) },
  inv := { app := λ F x, { app := λ X a, (F.1.map a) x.down } } }.
```

How does this work?

- an approximation of Ganesalingam-Gowers 'human-style automation' in Lean (arXiv:1309.4501)

- an algorithm for automatic rewriting, using an edit distance heuristic and some machine learning.

(in progress, w/ Keeley Hoek, ANU)