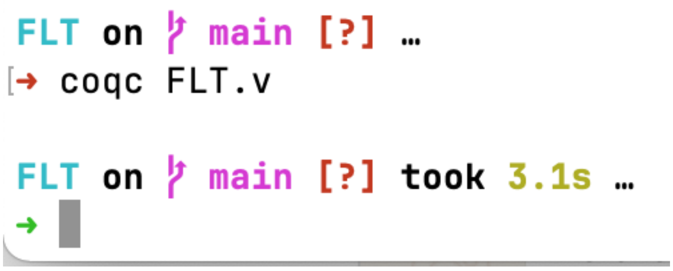
```
From Coq Require Import Arith Lia Reals ZArith Ring Lra.

(* ============================================================== *)
(*  GN(2) route to FLT (conditional, final formulation).         *)
(*                                                              *)
(*  Core hypothesis (GN(2)): for any n>2 and x,y,z ∈ ℕ,          *)
(*      x^n + y^n = z^n  ⇒  2^n = 2·n.                           *)
(*  Together with the elementary growth fact                     *)
(*      2^n = 2·n  ⇒  n ∈ {1,2},                                 *)
(*  this yields an immediate contradiction for n>2 (FLT).         *)
(*                                                              *)
(*  Implementation notes:                                        *)
(*  - The core proof is purely over naturals (no parameter o).    *)
(*  - An optional real "wrapper" uses                            *)
(*        covers_with 2 n  :=  pow 2 n = 2 * INR n               *)
(*    and bridge lemmas to recover 2^n = 2·n in ℕ.                *)
(*  - Identities over ℝ/ℤ (parameterization and parity) serve     *)
(*    only as motivation and are not used in the final step.      *)
(*                                                              *)
(*  Legacy "o>1, maximum coverage" formulation is retired.        *)
(* ============================================================== *)


(* ---------- Real-parameter identities (m,p ∈ R) ---------- *)
Local Open Scope R_scope.

Lemma sum_diff_from_parameters_R
      (n : nat) (m p : R) :
  let z := pow m n + pow p n in
  let x := pow m n - pow p n in
  z + x = 2 * pow m n /\
  z - x = 2 * pow p n.
Proof.
  intros z x; unfold z, x; split; ring.
Qed.

Close Scope R_scope.

(* ---------- Integer-parameter specialisation (m,p ∈ Z) ---------- *)
Local Open Scope Z_scope.

Lemma sum_diff_from_parameters_Z
      (n : nat) (m p : Z) :
  let z := m ^ Z.of_nat n + p ^ Z.of_nat n in
  let x := m ^ Z.of_nat n - p ^ Z.of_nat n in
  z + x = 2 * m ^ Z.of_nat n /\
  z - x = 2 * p ^ Z.of_nat n.
Proof.
  intros z x; unfold z, x; split; nia.
Qed.

Corollary parity_condition_Z
          (n : nat) (m p : Z) :
  let z := m ^ Z.of_nat n + p ^ Z.of_nat n in
  let x := m ^ Z.of_nat n - p ^ Z.of_nat n in
  Z.even (z + x) = true /\
  Z.even (z - x) = true.
Proof.
  intros z x.
  destruct (sum_diff_from_parameters_Z n m p) as [Hzx Hzx'].
  split.
```

```
  - replace (z + x) with (2 * m ^ Z.of_nat n) by exact Hzx.
    rewrite Z.even_mul; simpl; reflexivity.
  - replace (z - x) with (2 * p ^ Z.of_nat n) by exact Hzx'.
    rewrite Z.even_mul; simpl; reflexivity.
Qed.

Lemma no_parameters_if_parity_violation (n : nat) (z x : Z) :
  Z.even (z + x) = false \/ Z.even (z - x) = false ->
  ~ (exists m p : Z,
        z = m ^ Z.of_nat n + p ^ Z.of_nat n /\
        x = m ^ Z.of_nat n - p ^ Z.of_nat n).
Proof.
  intros Hpar [m [p [Hz Hx]]].
  destruct (sum_diff_from_parameters_Z n m p) as [Hsum Hdiff].
  destruct Hpar as [H1|H2].
  - rewrite Hz, Hx, Hsum in H1.
    rewrite Z.even_mul in H1; simpl in H1. discriminate.
  - rewrite Hz, Hx, Hdiff in H2.
    rewrite Z.even_mul in H2; simpl in H2. discriminate.
Qed.

Lemma no_parameters_if_odd (n : nat) (z x : Z) :
  Z.odd (z + x) = true \/ Z.odd (z - x) = true ->
  ~ (exists m p : Z,
        z = m ^ Z.of_nat n + p ^ Z.of_nat n /\
        x = m ^ Z.of_nat n - p ^ Z.of_nat n).
Proof.
  intros Hodd [m [p [Hz Hx]]].
  destruct (sum_diff_from_parameters_Z n m p) as [Hsum Hdiff].
  destruct Hodd as [H1|H2].
  - rewrite Hz, Hx, Hsum in H1.
    rewrite Z.odd_mul in H1; simpl in H1. discriminate.
  - rewrite Hz, Hx, Hdiff in H2.
    rewrite Z.odd_mul in H2; simpl in H2. discriminate.
Qed.

Lemma no_parameters_for_example :
  ~ (exists m p : Z,
        2%Z = m ^ Z.of_nat 3 + p ^ Z.of_nat 3 /\
        1%Z = m ^ Z.of_nat 3 - p ^ Z.of_nat 3).
Proof.
  apply (no_parameters_if_parity_violation 3 2 1).
  now left.
Qed.

Close Scope Z_scope.

(* ---------- Elementary growth facts on naturals ---------- *)
Local Open Scope nat_scope.

Lemma pow2_gt_linear_shift (k : nat) :
  2 ^ (k + 3) > 2 * (k + 3).
Proof.
  induction k as [|k IH]; simpl.
  - lia.
  - replace (S k + 3) with (k + 4) by lia.
    replace (2 ^ (S k + 3)) with (2 * 2 ^ (k + 3)) by
        (replace (S k + 3) with (S (k + 3)) by lia;
         rewrite Nat.pow_succ_r; lia).
    assert (Htmp : 2 * 2 ^ (k + 3) > 4 * (k + 3)) by nia.
    apply Nat.le_lt_trans with (m := 4 * (k + 3)).
```

```
    + lia.
    + exact Htmp.
Qed.

Lemma pow2_gt_linear (n : nat) :
  3 <= n -> 2 ^ n > 2 * n.
Proof.
  intros Hn.
  destruct (Nat.le_exists_sub 3 n Hn) as [k [Hk _]].
  rewrite Hk.
  replace (3 + k) with (k + 3) by lia.
  apply pow2_gt_linear_shift.
Qed.

Lemma pow_eq_linear_cases (n : nat) :
  2 ^ n = 2 * n -> n = 0 \/ n = 1 \/ n = 2.
Proof.
  destruct n as [|n].
  - simpl. intro H. now left.
  - destruct n as [|n].
    + simpl. intro H. right; left; lia.
    + destruct n as [|n].
      * simpl. intro H. right; right; lia.
      * intro H.
        assert (3 <= S (S (S n))) by lia.
        specialize (pow2_gt_linear _ H0) as Hgt.
        rewrite H in Hgt. lia.
Qed.

Lemma pow_eq_linear_positive (n : nat) :
  2 ^ n = 2 * n -> n = 1 \/ n = 2.
Proof.
  intro H.
  destruct (pow_eq_linear_cases n H) as [H0 | [H1 | H2]].
  - subst n. discriminate.
  - now left.
  - now right.
Qed.

Lemma pow3_gt_linear_shift (k : nat) :
  3 ^ (k + 1) > 2 * (k + 1).
Proof.
  induction k as [|k IH]; simpl.
  - lia.
  - replace (S k + 1) with (k + 2) by lia.
    replace (3 ^ (S k + 1)) with (3 * 3 ^ (k + 1)) by
        (replace (S k + 1) with (S (k + 1)) by lia;
         rewrite Nat.pow_succ_r; lia).
    assert (Htmp : 3 * 3 ^ (k + 1) > 3 * (2 * (k + 1))) by nia.
    apply Nat.le_lt_trans with (m := 3 * (2 * (k + 1))).
    + lia.
    + exact Htmp.
Qed.

Lemma pow3_gt_linear (n : nat) :
  1 <= n -> 3 ^ n > 2 * n.
Proof.
  intros Hn.
  destruct (Nat.le_exists_sub 1 n Hn) as [k [Hk _]].
  rewrite Hk.
  replace (1 + k) with (k + 1) by lia.
```

```
  apply pow3_gt_linear_shift.
Qed.

Lemma covers_two_nat (n : nat) :
  pow 2 n = INR (2 ^ n).
Proof.
  rewrite pow_INR.
  reflexivity.
Qed.

Lemma INR_two_mul_nat (n : nat) :
  (2 * INR n)%R = INR (2 * n).
Proof.
  rewrite mult_INR.
  simpl.
  reflexivity.
Qed.

(* ------------------------------------------------------------- *)
(*  GN(2) ⇒ FLT over naturals, with a real wrapper for coverage  *)
(* ------------------------------------------------------------- *)

Definition covers_with (o : R) (n : nat) := pow o n = (2 * INR n)%R.

Definition GN2 : Prop :=
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n = Nat.pow z n) ->
    2 ^ n = 2 * n.

(* ---------- GN(2) core, purely over naturals ---------- *)
Lemma FLT_from_GN2 :
  GN2 ->
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n = Nat.pow z n) -> False.
Proof.
  intros HGN2 n x y z Hn Heq.
  specialize (HGN2 n x y z Hn Heq) as Hcover.
  apply pow_eq_linear_positive in Hcover.
  lia.
Qed.

Definition GN2_R :=
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n = Nat.pow z n) ->
    pow 2 n = (2 * INR n)%R.

Lemma GN2_R_implies_GN2 : GN2_R -> GN2.
Proof.
  intros H n x y z Hn Heq.
  specialize (H n x y z Hn Heq).
  rewrite covers_two_nat in H.
  rewrite INR_two_mul_nat in H.
  apply INR_eq in H.
  exact H.
Qed.

Corollary fermat_last_theorem_from_GN2_R :
  GN2_R ->
```

```
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n = Nat.pow z n) -> False.
Proof.
  intros HGN2R.
  apply FLT_from_GN2.
  apply GN2_R_implies_GN2.
  exact HGN2R.
Qed.
```



Code available at:
https://github.com/Gendalf71/FLT-Coq