```
From Coq Require Import Init.Prelude.
From Coq Require Import Arith Lia Reals ZArith Ring Lra.
From Coq Require Import Arith.PeanoNat.
Module GlobalNormalization.
(* ============================================================ *)
(*  Global-normalization reading of Dedenko's manuscript.       *)
(*  Parameters m,p range over the reals; parity arguments are   *)
(*  recovered by specialising to integers.  A single real factor *)
(*  o>1 is postulated to serve all putative counterexamples, and *)
(*  the principle of maximum coverage selects the unique choice  *)
(*  o = 2, restricting the admissible exponents to n ∈ {1,2}.    *)
(* ============================================================ *)

(* ---------- Auxiliary binomial divisibility facts ---------- *)
Local Open Scope Z_scope.

#[local] Lemma Z_sub_split (a b c : Z) : a - b = (a - c) + (c - b).
Proof. ring. Qed.

#[local] Lemma Zpow_diff_divides (x y : Z) (n : nat) :
  Z.divide (x - y) (Z.pow x (Z.of_nat n) - Z.pow y (Z.of_nat n)).
Proof.
  induction n as [|n IH].
  - simpl. exists 0%Z; ring.
  - replace (Z.of_nat (S n)) with (Z.of_nat n + 1) by lia.
    rewrite !Z.pow_add_r by lia.
    rewrite !Z.pow_1_r.
    set (ux := Z.pow x (Z.of_nat n)) in *.
    set (uy := Z.pow y (Z.of_nat n)) in *.
    specialize (IH) as [q Hq].
    rewrite Z.mul_comm with (n := ux) (m := x).
    rewrite Z.mul_comm with (n := uy) (m := y).
    rewrite (Z_sub_split (x * ux) (y * uy) (x * uy)).
    assert (Hx : x * ux - x * uy = x * (ux - uy)) by ring.
    assert (Hy : x * uy - y * uy = (x - y) * uy) by ring.
    rewrite Hx, Hy.
    rewrite Hq.
    exists (x * q + uy).
    ring.
Qed.

#[local] Lemma odd_n_diff_pow_even
      (a b : Z) (n : nat) :
  Nat.odd n = true ->
  Z.divide 2 (Z.pow (a + b) (Z.of_nat n) - Z.pow (a - b) (Z.of_nat n)).
Proof.
  intros Hodd.
  apply Nat.odd_spec in Hodd.
  destruct Hodd as [k Hk].
  subst n.
  replace (Z.of_nat (2 * k + 1)) with (Z.of_nat (S (2 * k))) by lia.
  specialize (Zpow_diff_divides (a + b) (a - b) (S (2 * k))) as Hdiv.
  replace ((a + b) - (a - b)) with (2 * b) in Hdiv by ring.
  destruct Hdiv as [q Hq].
  exists (b * q).
  rewrite Hq.
  ring.
Qed.

Close Scope Z_scope.

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

#[local] 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; ring.
Qed.

#[local] 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.

#[local] 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.

#[local] 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.

#[local] 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.

#[local] 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 by lia;
         rewrite Nat.mul_comm; reflexivity).
    assert (IH' : 2 ^ (k + 3) > 2 * (k + 3)) by exact IH.
    assert (Htmp : 2 * 2 ^ (k + 3) > 2 * (2 * (k + 3))).
    { apply Nat.mul_lt_mono_pos_l; [lia|exact IH']. }
    replace (2 * (2 * (k + 3))) with (4 * (k + 3)) in Htmp by lia.
    apply Nat.le_lt_trans with (m := 4 * (k + 3)).
    + lia.
```

```
    + exact Htmp.
Qed.

#[local] 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.

#[local] 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.

#[local] 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.

#[local] Lemma pow_INR_natpow (k n : nat) :
  pow (INR k) n = INR (k ^ n).
Proof.
  rewrite pow_INR.
  reflexivity.
Qed.

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

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

#[local] Lemma pow_pow_mul : forall (a : R) (n m : nat),
  pow (pow a n) m = pow a (n * m).
Proof.
  intros a n m.
  induction m as [|m IH]; simpl.
  - now rewrite Nat.mul_0_r.
  - rewrite IH.
    rewrite Nat.mul_succ_r.
    rewrite pow_add.
    simpl.
    rewrite Rmult_comm.
    reflexivity.
Qed.

(* ------------------------------------------------------------- *)
```

```
(*  Global normalisation and Fermat's Last Theorem               *)
(* ------------------------------------------------------------ *)
Local Open Scope R_scope.

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

(* PUBLIC *)
Lemma covers_with_one_forces_two (o : R) :
  covers_with o 1%nat -> o = 2.
Proof.
  unfold covers_with.
  simpl.
  intro H.
  lra.
Qed.

#[local] Lemma INR_inj_nat (a b : nat) : INR a = INR b -> a = b.
Proof.
  intro H; first [apply INR_eq in H | apply eq_INR in H | apply INR_inj in H];
  exact H.
Qed.

(* PUBLIC: the "bridge" *)
Lemma two_real_normalizations_imply_nat_power_eq
      (o : R) (n m : nat) :
  covers_with o n ->
  covers_with o m ->
  Nat.pow (2 * n) m = Nat.pow (2 * m) n.
Proof.
  intros Hn Hm.
  unfold covers_with in *.
  rewrite INR_two_mul_nat in Hn.
  rewrite INR_two_mul_nat in Hm.
  assert (Hleft : pow (INR (2 * n)) m = pow o (n * m)).
  { rewrite <- Hn. apply pow_pow_mul. }
  assert (Hright : pow (INR (2 * m)) n = pow o (n * m)).
  { rewrite <- Hm. rewrite Nat.mul_comm. apply pow_pow_mul. }
  rewrite <- Hleft in Hright.
  apply INR_inj_nat.
  pose proof (pow_INR_natpow (2 * n) m) as Hpow_n.
  pose proof (pow_INR_natpow (2 * m) n) as Hpow_m.
  rewrite <- Hpow_n.
  rewrite <- Hpow_m.
  symmetry.
  exact Hright.
Qed.

(* PUBLIC *)
Lemma covers_with_two_characterisation (n : nat) :
  covers_with 2 n -> n = 1%nat \/ n = 2%nat.
Proof.
  unfold covers_with.
  intro H.
  rewrite covers_two_nat in H.
  rewrite INR_two_mul_nat in H.
  apply INR_inj_nat in H.
  apply pow_eq_linear_positive in H.
  assumption.
Qed.

(* PUBLIC *)
Lemma maximum_coverage_as_theorem
      (o : R) :
  covers_with o 1%nat ->
  (forall n, covers_with o n -> n = 1%nat \/ n = 2%nat) /\ o = 2.
Proof.
  intro H1.
  pose proof (covers_with_one_forces_two o H1) as Ho.
  split; [|exact Ho].
  intros n Hn.
  subst o.
  apply covers_with_two_characterisation.
  exact Hn.
Qed.
```

```
Section Global_Normalization.

Variable o : R.
Hypothesis normalization_equation :
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n ->
    covers_with o n.
Hypothesis coverage_one : covers_with o 1%nat.

(* PUBLIC *)
Lemma normalization_parameter_is_two : o = 2.
Proof.
  destruct (maximum_coverage_as_theorem o coverage_one) as [_ Ho].
  exact Ho.
Qed.

(* PUBLIC *)
Lemma normalization_forces_small_exponent :
    forall n x y z,
      (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n -> False.
Proof.
  intros n x y z Hn Heq.
  specialize (normalization_equation n x y z Hn Heq) as Hcover.
  destruct (maximum_coverage_as_theorem o coverage_one)
    as [Hcov Ho].
  specialize (Hcov n Hcover) as [Hn1 | Hn2]; lia.
Qed.

End Global_Normalization.

#[local] Lemma covers_two_one : covers_with 2 1%nat.
Proof.
  unfold covers_with; simpl.
  lra.
Qed.

#[local] Lemma covers_two_two : covers_with 2 2%nat.
Proof.
  unfold covers_with; simpl.
  lra.
Qed.

#[local] Lemma covers_two_only_small (n : nat) :
  covers_with 2 n -> n = 1%nat \/ n = 2%nat.
Proof.
  apply covers_with_two_characterisation.
Qed.

(* PUBLIC *)
Corollary fermat_last_theorem_from_global_normalization :
  (forall (n x y z : nat),
      (2 < n)%nat ->
      (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n ->
      covers_with 2 n) ->
  forall (n x y z : nat),
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n -> False.
Proof.
  intros Hnorm n x y z Hn Heq.
  specialize (Hnorm n x y z Hn Heq) as Hcover.
  assert (covers_with 2 1%nat) as Hone by apply covers_two_one.
  specialize (maximum_coverage_as_theorem 2 Hone)
    as [Hrest _].
  specialize (Hrest n Hcover) as [Hn1 | Hn2]; lia.
Qed.

(* PUBLIC *)
Corollary fermat_last_theorem_via_maximum_coverage :
  (forall (n x y z : nat),
      (2 < n)%nat ->
      (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n ->
      pow 2 n = 2 * INR n) ->
  forall (n x y z : nat),
```

```
    (2 < n)%nat ->
    (Nat.pow x n + Nat.pow y n)%nat = Nat.pow z n -> False.
Proof.
  intros Hnorm n x y z Hn Heq.
  apply (fermat_last_theorem_from_global_normalization Hnorm n x y z Hn Heq).
Qed.

(* ---------- Minimal p-adic bracket for the universal parameter ---------- *)
Section PadicBracket.

Local Open Scope Z_scope.

Record OPadic := { vp_o : nat -> Z }.

#[local] Definition NatOddGreater1 (p : nat) : Prop := (1 < p)%nat.

#[local] Definition padic_equation (o : OPadic) (n : nat) : Prop :=
  forall p, NatOddGreater1 p -> Nat.odd p = true -> Z.of_nat n * vp_o o p = 0%Z.

#[local] Lemma odd_primes_vanish_in_o
      (o : OPadic) :
  padic_equation o 1%nat ->
  padic_equation o 2%nat ->
  forall p, NatOddGreater1 p -> Nat.odd p = true -> vp_o o p = 0%Z.
Proof.
  intros H1 H2 p Hp Hodd.
  specialize (H1 p Hp Hodd).
  change (Z.of_nat 1) with 1%Z in H1.
  rewrite Z.mul_1_l in H1.
  exact H1.
Qed.

#[local] Lemma two_adic_normalisation (o : OPadic) :
  Z.of_nat 1 * vp_o o 2%nat = 1%Z -> vp_o o 2%nat = 1%Z.
Proof.
  intro H.
  change (Z.of_nat 1) with 1%Z in H.
  now rewrite Z.mul_1_l in H.
Qed.

End PadicBracket.

(* ---------- Sanity goals for quick regression checks ---------- *)

Goal covers_with 2 3%nat -> False.
Proof.
  intro H.
  destruct (covers_two_only_small 3%nat H) as [H1|H2]; lia.
Qed.

Goal forall n : nat, (2 < n)%nat -> covers_with 2 n -> False.
Proof.
  intros n Hn Hcover.
  apply covers_two_only_small in Hcover as [H1|H2]; lia.
Qed.
End GlobalNormalization.
Export GlobalNormalization.
```

```
FLT on  main …
[➜ coqc GlobalNormalization.v

FLT on  main [?] took 2.3s …
➜
```