```
import Mathlib

open Finsupp

open scoped Finset

variable {R : Type*} [CommRing R]

open MvPolynomial

/-- Lemma 2.2 :
A multivariate polynomial that vanishes on a large product finset is the zero
    polynomial. -/
lemma eq_zero_of_eval_zero_at_prod_finset {σ : Type*} [Finite σ] [IsDomain R]
    (P : MvPolynomial σ R) (S : σ → Finset R)
    (Hdeg : ∀ i, P.degreeOf i < #(S i))
    (Heval : ∀ (x : σ → R), (∀ i, x i ∈ S i) → eval x P = 0) :
    P = 0 := by
      exact MvPolynomial.eq_zero_of_eval_zero_at_prod_finset P S Hdeg Heval

variable {p : ℕ} [Fact (Nat.Prime p)] {k : ℕ}

/-- Definition of elimination polynomials g_i -/
noncomputable def elimination_polynomials (A : Fin (k + 1) → Finset (ZMod p)) :
    Fin (k + 1) → MvPolynomial (Fin (k + 1)) (ZMod p) :=
  fun i => ∏ a ∈ A i, (MvPolynomial.X i - C a)

noncomputable def reduce_polynomial_degrees (P : MvPolynomial (Fin (k + 1)) (
    ZMod p))
    (g : Fin (k + 1) → MvPolynomial (Fin (k + 1)) (ZMod p))
    (c : Fin (k + 1) → ℕ) : MvPolynomial (Fin (k + 1)) (ZMod p) :=

  P.support.sum fun m =>
    let coeff := P.coeff m
    let needs_replacement : Finset (Fin (k + 1)) :=
      Finset.filter (fun i => m i > c i) Finset.univ
    if h : needs_replacement.Nonempty then
      let i : Fin (k + 1) := needs_replacement.min' h
      let new_m : (Fin (k + 1)) →₀ ℕ :=
        Finsupp.update m i (m i - (c i + 1))
      coeff • (MvPolynomial.monomial new_m 1) * g i
    else
      coeff • MvPolynomial.monomial m 1

set_option maxHeartbeats 2000000 in
/-- **Alon-Nathanson-Ruzsa Theorem** (Theorem 2.1)
Proof strategy: Use Lemma 2.2 (eq_zero_of_eval_zero_at_prod_finset) to prove
    Theorem 2.1
```

```
Proof outline:
1. Assume the conclusion is false, i.e., there exists a set E subset Z_p with |
   E| = m such that restricted sumset subset E
2. Construct the polynomial Q(x_0,...,x_k) = h(x_0,...,x_k) * prod_{e in E} (
   x_0+...+x_k - e)
   - deg(Q) = deg(h) + m = sum c_i
   - For all (a_0,...,a_k) in prod A_i, we have Q(a_0,...,a_k) = 0
   - The coefficient of monomial prod x_i^{c_i} in Q is nonzero

3. For each i, define g_i(x_i) = prod_{a in A_i} (x_i - a) = x_i^{c_i+1} -
   sum_j b_ij x_i^j
4. Construct polynomial Q_bar by replacing all occurrences of x_i^{c_i+1} in Q
   with sum_j b_ij x_i^j
   - For each a_i in A_i, g_i(a_i) = 0, so Q_bar still vanishes on prod A_i
   - deg_{x_i}(Q_bar) <= c_i

5. Apply Lemma 2.2:
   - Q_bar vanishes on prod A_i
   - Degree in each variable <= c_i
   - Therefore Q_bar = 0

6. But the coefficient of prod x_i^{c_i} in Q_bar is the same as in Q:
   - The replacement process doesn't affect this specific monomial
   - By assumption, this coefficient is nonzero in Q
   - Therefore it's nonzero in Q_bar, contradicting Q_bar = 0

Key points:
- Use polynomial replacement technique to reduce degrees to satisfy Lemma 2.2
  conditions
- The replacement process preserves the coefficient of the target monomial
- Proof by contradiction
-/
theorem ANR_polynomial_method (h : MvPolynomial (Fin (k + 1)) (ZMod p))
    (A : Fin (k + 1) → Finset (ZMod p))
    (c : Fin (k + 1) → ℕ)
    (hA : ∀ i, (A i).card = c i + 1)
    (m : ℕ) (hm : m = (∑ i, c i) - h.totalDegree)
    (h_coeff : MvPolynomial.coeff (Finsupp.equivFunOnFinite.symm c)
    ((∑ i : Fin (k + 1), MvPolynomial.X i) ^ m * h) ≠ 0) :
    let S : Finset (ZMod p) :=
      (Fintype.piFinset A).filter (fun f => h.eval f ≠ 0) |>.image (fun f => ∑
        i, f i)
    S.card ≥ m + 1 ∧ m < p := by
  -- Define the restricted sumset S
  set S : Finset (ZMod p) :=
  ((Fintype.piFinset A).filter (fun f => h.eval f ≠ 0)).image (fun f => ∑ i, f
     i) with hS_def

  -- Step 1: Prove |S| >= m + 1 by contradiction
```

```
    have hS_card : S.card ≥ m + 1 := by
      by_contra! H

      have hS_size : S.card ≤ m := by omega
      obtain ⟨E, hE_sub, hE_card⟩ : ∃ E : Multiset (ZMod p), S.val ⊆ E ∧ E.card =
          m := by
        refine ⟨S.val + Multiset.replicate (m - S.card) (0 : ZMod p),
                Multiset.subset_of_le (by simp), ?_⟩
        simp [hS_size]

      -- Define the polynomial Q
      set sumX : MvPolynomial (Fin (k + 1)) (ZMod p) := ∑ i, MvPolynomial.X i
        with hsumX_def
      set Q : MvPolynomial (Fin (k + 1)) (ZMod p) :=
        h * (E.map (fun e => sumX - C e)).prod with hQ_def

      -- Q vanishes on prod A_i
      have hQ_zero : ∀ (x : Fin (k + 1) → ZMod p), (∀ i, x i ∈ A i) → eval x Q =
          0 := by
        intro x hx
        rw [hQ_def, eval_mul]
        by_cases hh : eval x h = 0
        · simp [hh]
        · have h_sum_in_S : (∑ i, x i) ∈ S := by
            simp [S, Fintype.mem_piFinset]
            refine ⟨x, ⟨hx, hh⟩, rfl⟩
          have h_sum_in_E : (∑ i, x i) ∈ E := hE_sub h_sum_in_S
          have : eval x ((E.map (fun e => sumX - C e)).prod) = 0 := by
            have mem : (sumX - C (∑ i, x i)) ∈ Multiset.map (fun e => sumX - C e)
                 E :=
              Multiset.mem_map.mpr ⟨∑ i, x i, h_sum_in_E, rfl⟩
            have zero_eval : eval x (sumX - C (∑ i, x i)) = 0 := by
              simp [hsumX_def]
            have hprod_eq_zero : (MvPolynomial.eval x) (sumX - C (∑ i, x i)) = 0
                := by exact
              zero_eval
            have eval_factor_zero : eval x (sumX - C (∑ i, x i)) = 0 := by exact
                zero_eval
            have prod_zero :
            (MvPolynomial.eval x) (Multiset.map (fun e => sumX - C e) E).prod = 0
                 := by
              have : (MvPolynomial.eval x) ((Multiset.map (fun e => sumX - C e) E
                  ).prod) =
                  (Multiset.map (fun e => (MvPolynomial.eval x) (sumX - C e)) E
                      ).prod := by
                  exact Eq.symm (Multiset.prod_hom' E (MvPolynomial.eval x) fun
                       i ↦ sumX - C i)
              rw [this]
              subst hm
```

```
            simp_all [S, sumX, Q]
            obtain ⟨w, h_1⟩ := h_sum_in_S
            obtain ⟨w_1, h_2⟩ := mem
            obtain ⟨left, right⟩ := h_1
            obtain ⟨left_1, right_1⟩ := h_2
            obtain ⟨left, right_2⟩ := left
            apply Exists.intro
            • apply And.intro
              • apply h_sum_in_E
              • simp_all only [sub_self]
          subst hm
          simp_all [S, sumX, Q]
        simp only [this, mul_zero]

    have hQ_total_deg : Q.totalDegree = ∑ i, c i := by
      rw [hQ_def, hsumX_def]
      have h_prod_deg : ((E.map (fun e => sumX - C e)).prod).totalDegree = m :=
          by
        rw [hsumX_def]
        have degree_of_each : ∀ e : ZMod p, (sumX - C e).totalDegree = 1 := by
          intro e
          rw [hsumX_def]
          have : (∑ i : Fin (k + 1), X i - C e) = (∑ i, X i) + (-C e) := by rw
              [sub_eq_add_neg]
          rw [MvPolynomial.totalDegree]
          apply le_antisymm
          • refine Finset.sup_le fun d hd => ?_
            simp [this] at hd
            have : (d.sum fun x e ↦ e) ≤ 1 := by
              -- Step 1: Coefficient decomposition of sum X_i - C e
              have coeff_sub_eq :
              coeff d (∑ i, X i - C e) = coeff d (∑ i, X i) + -coeff d (C e) :=
                  by
                simp [MvPolynomial.coeff_sub]
                exact sub_eq_add_neg (coeff d (∑ i, X i)) (if 0 = d then e else
                    0)
              have coeff_C_eq : coeff d (C e) = if d = 0 then e else 0 := by
                simp [MvPolynomial.coeff_C]
                subst hE_card
                simp_all [S, sumX, Q]
                split
                next h_1 =>
                  subst h_1
                  simp_all only [↓reduceIte]
                next h_1 =>
                  simp_all only [↓reduceIte, neg_zero, add_zero,
                      right_eq_ite_iff]
                  intro a
                  subst a
```

```
        simp_all only [not_true_eq_false]
    rw [coeff_C_eq] at coeff_sub_eq
    -- Step 2: d is in the support since its coefficient != 0
    have mem_support : d ∈ (∑ i, X i - C e).support := by
      rw [MvPolynomial.mem_support_iff, coeff_sub_eq]
      subst hE_card
      simp_all [ S, sumX, Q]
    have support_subset :
    (∑ i, X i - C e).support ⊆
        (Finset.biUnion Finset.univ fun i : Fin (k + 1) => {Finsupp.
            single i 1})
        ∪ {0} := by
      intro x hx
      rw [MvPolynomial.mem_support_iff] at hx
      have coeff_eq : coeff x (∑ i, X i - C e) = coeff x (∑ i, X i) -
          coeff x (C e) := by
        simp [MvPolynomial.coeff_sub]
      rw [coeff_eq] at hx
      have coeff_sum_eq :
      coeff x (∑ i, X i) = if ∃ i, x = Finsupp.single i 1 then 1 else
          0 := by
        simp [MvPolynomial.coeff_sum]
        have h :
        ∑ x_1, coeff x (X x_1) = if ∃ i, x = Finsupp.single i 1 then
            1 else 0 := by
          by_cases h : ∃ i, x = Finsupp.single i 1
          · rcases h with ⟨i, hi⟩
            have :
            ∀ j, (if x = Finsupp.single j 1
            then (1 : ZMod p) else 0) = if i = j then 1 else 0
            :=
            by
              intro j
              rw [hi]
              by_cases h : i = j
              · subst h
                simp only [↓reduceIte]
              · simp [h]
                rw [@single_eq_single_iff]
                ring_nf
                simp_all
            aesop
          · push_neg at h
            simp [h]
            have : ∀ i, coeff x (X i) = 0 := by
              intro i
              simp [MvPolynomial.coeff_X']
              exact fun a ↦ h i (id (Eq.symm a))
            exact fun i => this i
```

```
          subst hm
          simp_all [S, sumX, Q]
        have coeff_C_eq : coeff x (C e) = if x = 0 then e else 0 := by
          simp [MvPolynomial.coeff_C]
          subst hm
          simp_all [S, sumX, Q]
          split
          next h_1 =>
            subst h_1
            simp_all only [↓reduceIte]
          next h_1 =>
            simp_all only [↓reduceIte, sub_zero, neg_zero, add_zero,
                right_eq_ite_iff]
            intro a
            subst a
            simp_all only [not_true_eq_false]
        by_cases h : ∃ i, x = Finsupp.single i 1
        · rcases h with ⟨i, hi⟩
          apply Finset.mem_union_left
          simp [hi]
          use i
        · rw [Finset.mem_union, Finset.mem_singleton]
          right
          have h1 : coeff x (∑ i, X i) = (0 : ZMod p) := by
            have : ¬∃ i, x = Finsupp.single i 1 := h
            sorry
          rw [h1] at hx
          rw [coeff_C_eq] at hx
          by_cases h2 : x = 0
          · exact h2
          · simp [h2] at hx
      have :
      d ∈ (Finset.biUnion Finset.univ fun i : Fin (k + 1) => {Finsupp.
          single i 1}) ∪ {0} :=
        support_subset mem_support
      simp at this
      subst hm
      simp_all only [S, sumX, Q]
      cases this with
      | inl h_1 =>
        subst h_1
        simp_all only [↓reduceIte, sum_zero_index, zero_le]
      | inr h_2 =>
        obtain ⟨w, h_1⟩ := h_2
        subst h_1
        simp_all only [single_eq_zero, one_ne_zero, ↓reduceIte,
            neg_zero, add_zero,
          sum_single_index, le_refl]
    exact this
```

```
        simp
        let b : (Fin (k + 1)) →₀ ℕ := Finsupp.single (0 : Fin (k + 1)) 1
        refine ⟨b, ?_, ?_⟩
        · have coeff_eq : coeff (Finsupp.single (0 : Fin (k + 1)) 1) (∑ i, X
            i) = 1 := by
            simp [coeff_sum, coeff_X', Finsupp.single_eq_single_iff]
          rw [show b = Finsupp.single (0 : Fin (k + 1)) 1 by rfl] at *
          have h : coeff b (∑ i, X i) = 1 := by
            dsimp [b]
            exact coeff_eq
          sorry
        · simp only [sum_single_index, le_refl, b]
      · sorry
    have h_h_deg : h.totalDegree = (∑ i, c i) - m := by
      rw [hm]

      sorry
    sorry
  --
  have hQ_coeff : MvPolynomial.coeff (Finsupp.equivFunOnFinite.symm c) Q ≠ 0
      := by
    rw [hQ_def, coeff_mul]
    sorry

  -- Define elimination polynomials g_i
  set g : Fin (k + 1) → MvPolynomial (Fin (k + 1)) (ZMod p) :=
    elimination_polynomials A with hg_def

  -- Construct Q_bar by reducing degrees
  set Q_bar : MvPolynomial (Fin (k + 1)) (ZMod p) :=
    reduce_polynomial_degrees Q g c with hQ_bar_def

  -- Now the key properties of Q_bar
  have ⟨R, decomp⟩ :
    ∃ R, (E.map (fun e => sumX - C e)).prod = sumX^m + R :=
  by
    refine ⟨(E.map (fun e => sumX - C e)).prod - sumX^m, ?_⟩
    simp

  let target := Finsupp.equivFunOnFinite.symm c
  let P := (E.map (fun e => sumX - C e)).prod

  -- Q = h * P
  have coeff_decomp :
    (Q.coeff target) =
      (h * sumX^m).coeff target + (h * R).coeff target :=
  by sorry

  -- Q_bar vanishes on prod A_i
```

```
    have hQ_bar_zero : ∀ (x : Fin (k + 1) → ZMod p), (∀ i, x i ∈ A i) → eval x
        Q_bar = 0 := by
      intro x hx
      sorry

    -- Q_bar has degree constraints
    have hQ_bar_deg : ∀ i, Q_bar.degreeOf i ≤ c i := by
      intro i
      sorry

    have hQ_bar_zero_poly : Q_bar = 0 :=
      _root_.eq_zero_of_eval_zero_at_prod_finset Q_bar A (fun i => by
        have := hQ_bar_deg i
        grind) hQ_bar_zero

    -- But the coefficient of the target monomial in Q_bar is nonzero
    have hQ_bar_coeff : MvPolynomial.coeff (Finsupp.equivFunOnFinite.symm c)
        Q_bar ≠ 0 := by
      -- Note that the replacement process does not affect this coefficient
      sorry

    -- Contradiction
    rw [hQ_bar_zero_poly] at hQ_bar_coeff
    simp at hQ_bar_coeff

  -- Step 2: Prove m < p first (this is needed for the main argument)
  have hmp : m < p := by
    by_contra! H  -- H: m >= p
    -- If m >= p, use the Frobenius endomorphism property in characteristic p
    have frobenius_identity : ((∑ i : Fin (k + 1), MvPolynomial.X i) ^ p :
    MvPolynomial (Fin (k + 1)) (ZMod p)) = ∑ i, MvPolynomial.X i ^ p := by
      subst hm
      simp_all only [ne_eq, ge_iff_le, S]
      exact sum_pow_char p Finset.univ X

    -- This changes the structure of (sum X_i)^m when m >= p, leading to
        contradiction with h_coeff
    subst hm
    simp_all only [ne_eq, ge_iff_le, S]
    sorry  -- Detailed argument needed here


  exact ⟨hS_card, hmp⟩
```