Formalisation en logique de la conjecture de Goldbach en lean, recherchant une stratégie de démonstration, désespérément
Denise Vella-Chemla, 23 septembre 2026

```
_
import Mathlib

theorem even_number_property (n : ℕ) (h1 : 6 ≤ n) (h2 : n % 2 == 0) :
  ∃ p : ℕ, 3 ≤ p ∧ p ≤ n / 2 ∧
  ∀ q : ℕ, (3 ≤ q ∧ q ≤ ℕ.sqrt n) → (¬(p % q = 0) ∧ ¬(p % q = n % q)) := by
  sorry -- ou bien omega ou bien decide ou bien simp ou bien tauto qui refusent

def satisfies_condition (n p : ℕ) : Bool :=
  let sqrt_n := ℕ.sqrt n
  let qs := if sqrt_n ≥ 3 then List.range' 3 (sqrt_n - 3 + 1) else []
  qs.all (fun q => (p % q ≠ 0) && (p % q ≠ n % q))

def find_decomposants (n : ℕ) : List ℕ :=
  let max_p := n / 2
  let candidates := if max_p ≥ 3 then List.range' 3 (max_p - 3 + 1) else []
  candidates.filter (fun p => satisfies_condition n p)

def main : IO Unit := do
  let n : ℕ := 98
  IO.println s!"--- Test de la routine pour n = n ---"

  let resultats := find_decomposants n

  IO.println s!"Nombre n : n"
  IO.println s!"Racine carrée (sqrt n) : ℕ.sqrt n"
  IO.println s!"Limite supérieure (n / 2) : n / 2"
  IO.println s!"Décomposants valides trouvés : resultats"
  IO.println "----------------------------------------"

-- La logique pure (tauto) ne suffit pas.
-- La simplification algébrique (simp) ne trouve pas le témoin caché.
-- Le calcul brut (decide) refuse les variables générales.
-- L'arithmétique linéaire (omega) ignore tout des modulos
-- et de la structure multiplicative.
```