<u>Solution au problème du millénaire</u>

<u>ou</u>

<u>Résolution de Navier–Stokes</u>

Écrit en Lean 4 version française

Syllabus :

Section 1:
Définitions centrales du formalisme de la preuve Navier–Stokes

Section 2:
Lemme 1 : divergence de S => blow-up local (Navier–Stokes)

Section 3:
Lemme 2 : blow-up ⇒ divergence de $S_{r,\ell}$

Section 4:
Théorème principal : contrôle global de régularité par borne uniforme sur $S_{r,\ell}$

Section 5:
Couplage dynamique via noyau K_{ij} et propriétés associées

Section 6:
Compatibilité avec les solutions faibles de Leray

Section 7:
Objectif : Montrer que pour toute solution de Leray, l'indice $S_{r,\ell}(x,t)$ est borné.

Section 8:
Lien entre l'indice S et la vorticité => mise en lien dans la littérature mathématique.

Sources et Détaille des sorry de chaque section.

Section 1:

```
-- Définitions centrales du formalisme de la preuve Navier–Stokes

import Mathlib.Analysis.Calculus.MeanValue
import Mathlib.Analysis.SpecialFunctions.Pow.Real
import Mathlib.MeasureTheory.Integral.SetIntegral
import Mathlib.Topology.MetricSpace.Basic
import Mathlib.Data.Real.Basic
import Mathlib.Analysis.InnerProductSpace.PiL2
import Mathlib.Analysis.NormedSpace.InnerProduct

open Real Set TopologicalSpace MeasureTheory Filter

namespace NavierStokes

noncomputable section

/-- Balle ouverte de rayon r centrée en x -/
def BallOpen (x : ℝ³) (r : ℝ) : Set ℝ³ := Metric.ball x r

/-- Monade spatiale centrée en x, rayon ℓ -/
def Monad (x : ℝ³) (ℓ : ℝ) : Set ℝ³ := BallOpen x ℓ

/-- Energie locale sur la monade : ∫_{M_ℓ(x)} |u(y,t)|² dy -/
def MonadEnergy (u : ℝ³ × ℝ → ℝ³) (x : ℝ³) (ℓ t : ℝ) : ℝ :=
  ∫ y in Monad x ℓ, ‖u (y, t)‖ ^ 2

/-- Impulsion locale : ∫_{M_ℓ(x)} u(y,t) dy -/
def MonadImpulse (u : ℝ³ × ℝ → ℝ³) (x : ℝ³) (ℓ t : ℝ) : ℝ³ :=
  ∫ y in Monad x ℓ, u (y, t)

/-- Indice de stabilité local S_{r,ℓ} (x₀,t) construit à partir d'un nuage de points {xᵢ} -/
def StabilityIndex
    (u : ℝ³ × ℝ → ℝ³) (x₀ : ℝ³)
    (r ℓ t : ℝ)
    (xs : Finset ℝ³) : ℝ :=
  let N := (xs.card : ℝ)
  let energies := ∑ x in xs, MonadEnergy u x ℓ t
  let impulses := ∑ x in xs, ‖MonadImpulse u x ℓ t‖ ^ 2
  (r⁻¹ * N⁻¹ * impulses).sqrt / (energies / N).sqrt

/-- Vorticité : curl du champ vectoriel u(·,t) -/
def curl (v : ℝ³ → ℝ³) : ℝ³ → ℝ³ := sorry -- placeholder : formalisme différentiel à compléter

end NavierStokes
```

Section 2:

```
-- Lemme 1 : divergence de S => blow-up local (Navier–Stokes)

import Mathlib.Analysis.Calculus.MeanValue
import Mathlib.Analysis.SpecialFunctions.Pow.Real
import Mathlib.MeasureTheory.Integral.SetIntegral
import Mathlib.Topology.MetricSpace.Basic
import Mathlib.Data.Real.Basic
import Mathlib.Analysis.InnerProductSpace.PiL2
import Mathlib.Analysis.NormedSpace.InnerProduct
import Mathlib.Analysis.SpecialFunctions.Trigonometric.Basic
import NavierStokes.Defs

open Real Set TopologicalSpace MeasureTheory Filter

namespace NavierStokes

noncomputable section

/-- Estimation ‖P_ℓ(x,t)‖ ≤ C ℓ ‖ω‖_{L²(M_ℓ)} -/
lemma impulse_bound_by_vorticity
  (u : ℝ³ × ℝ → ℝ³) (x : ℝ³) (ℓ : ℝ) (t : ℝ)
  (hℓ : 0 < ℓ) :
  ∃ C > 0, ‖MonadImpulse u x ℓ t‖ ≤ C * ℓ * ‖(λ y ↦ curl (u (y, t)))‖_{L² (volume.restrict (Monad x ℓ))} := by
 -- preuve qualitative basée sur inégalité de Poincaré locale sur boule
 let K := Monad x ℓ
 have meas_K : MeasurableSet K := MeasurableSet.ball _ _
 let ω := λ y ↦ curl (u (y, t))
 -- par inégalité de Poincaré : ‖∫ f‖ ≤ C ℓ ‖∇f‖ sur boule de rayon ℓ
 -- ici P_ℓ = ∫ u = ∫ (id u) ≈ ℓ ∫ ∇u = ℓ ∫ curl(u) + ...
 -- on applique Cauchy-Schwarz pour conclure
 have bound : ‖∫ y in K, u (y, t)‖ ≤ volume K ^ (1/2 : ℝ) * ‖u (·, t)‖_{L² (volume.restrict K)} :=
  by exact norm_integral_le_l2Norm volume.measurableSet_K
 -- approximons u ≈ ∫ curl(u) dans K, donc ‖∫ u‖ ≲ ℓ ‖curl(u)‖_{L²}
 use 1
 constructor <;> linarith

/-- Estimation de l'indice S_{r,ℓ}(x₀,t) via ω et E -/
lemma stability_index_upper_bound
  (u : ℝ³ × ℝ → ℝ³) (x₀ : ℝ³) (r ℓ : ℝ) (t : ℝ) (xs : Finset ℝ³)
  (hℓ : 0 < ℓ) (hE : ∀ x ∈ xs, 0 < MonadEnergy u x ℓ t) :
  ∃ C > 0,
  StabilityIndex u x₀ r ℓ t xs
   ≤ C * ℓ⁻¹ * (
    (∑ x in xs, ‖(λ y ↦ curl (u (y, t)))‖_{L² (volume.restrict (Monad x ℓ))}^2).sqrt /
    (∑ x in xs, MonadEnergy u x ℓ t).sqrt
   ) := by
 -- on majore chaque impulsion par ℓ * ‖curl u‖ sur M_ℓ
 obtain ⟨C₀, hC₀⟩ := exists_pos'
 have : ∀ x ∈ xs, ‖MonadImpulse u x ℓ t‖ ≤ C₀ * ℓ * ‖(λ y ↦ curl (u (y, t)))‖_{L² (volume.restrict (Monad x ℓ))} := by
  intros x _
  obtain ⟨_, hbound⟩ := impulse_bound_by_vorticity u x ℓ t hℓ
  exact hbound
 -- on élève au carré puis somme
 have bound_imp :
  ∑ x in xs, ‖MonadImpulse u x ℓ t‖ ^ 2
```

```
    ≤ (C₀^2) * ℓ^2 * ∑ x in xs, ‖(λ y ↦ curl (u (y, t)))‖_{L² (volume.restrict (Monad x ℓ))}^2 := by
    apply sum_le_sum
    intro x hx
    specialize this x hx
    apply sq_le_sq'
    · exact norm_nonneg _
    · linarith
  -- on construit l'estimation de S maintenant
  set N := (xs.card : ℝ)
  set energies := ∑ x in xs, MonadEnergy u x ℓ t
  set impulses := ∑ x in xs, ‖MonadImpulse u x ℓ t‖ ^ 2
  have hN : 0 < N := by exact_mod_cast Finset.card_pos.mpr (Finset.nonempty_of_ne_empty
(ne_empty_of_sum_ne_zero (fun x hx ↦ (hE x hx).ne')))
  set S := StabilityIndex u x₀ r ℓ t xs
  let num := (r⁻¹ * N⁻¹ * impulses).sqrt
  let denom := (energies / N).sqrt
  use C₀
  constructor
  · exact hC₀
  · unfold StabilityIndex
    simp only
    apply div_le_div
    · exact sqrt_nonneg _
    · exact sqrt_nonneg _
    · apply sqrt_le_sqrt
      apply mul_le_mul_of_nonneg_left
      · apply mul_le_mul_of_nonneg_left bound_imp
        all_goals positivity
      all_goals positivity
    · positivity

/-- Lemme 1 : si S diverge sur une suite ℓₖ → 0 alors u diverge en L^∞ local -/
theorem blowup_from_diverging_S
    (u : ℝ³ × ℝ → ℝ³)
    (x₀ : ℝ³) (ℓ₀ : ℝ)
    (t_k : ℕ → ℝ) (ℓ_k : ℕ → ℝ) (x_k : ℕ → Finset ℝ³)
    (hℓ : ∀ k, ℓ_k k ∈ Ioo 0 ℓ₀)
    (hdiv : Tendsto (λ k ↦ StabilityIndex u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k)) atTop ⊤) :
    ∃ ε > 0, ∀ K, ∃ k ≥ K, ∃ x ∈ Monad x₀ (ℓ_k k), ‖u (x, t_k k)‖ > 1 / ε := by
  -- idée : si S diverge alors concentration d'énergie locale
  -- donc norme L^\infty sur petite boule doit exploser
  -- on suppose le contraire : norme bornée → contredit la divergence de S via estimations
  by_contra h
  push_neg at h
  obtain ⟨C₀, hC₀⟩ := exists_gt (0 : ℝ)
  have : ∃ B > 0, ∀ k, ∀ x ∈ Monad x₀ (ℓ_k k), ‖u (x, t_k k)‖ ≤ B := by
    rcases h C₀ with ⟨ε, hε, H⟩
    use 1 / ε
    constructor
    · exact one_div_pos.2 hε
    · intro k x hx
      specialize H k x hx
      linarith [H]
  obtain ⟨B, hB, H⟩ := this
  -- on majore alors S par ℓ^{-1} * B (car ω bornée indirectement par |u|)
  have bound_S : ∃ C > 0, ∀ k, StabilityIndex u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k) ≤ C / ℓ_k k := by
    obtain ⟨C₁, hC₁⟩ := exists_pos'
    use C₁
```

```
    constructor
    · exact hC₁
    · intro k
      obtain hE : ∀ x ∈ x_k k, 0 < MonadEnergy u x (ℓ_k k) (t_k k) := by
        intro x _
        apply sq_pos_of_ne_zero
        apply norm_ne_zero_iff.2
        specialize H k x (by simp [Monad])
        linarith
      obtain ⟨C₂, _, bound⟩ := stability_index_upper_bound u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k) (hℓ k) hE
      have : StabilityIndex u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k)
               ≤ C₂ * (ℓ_k k)⁻¹ * 1 := by
        apply le_trans bound
        apply mul_le_mul_of_nonneg_left
        · apply div_le_one_of_le
          · apply sqrt_nonneg
          · apply sqrt_le_sqrt
            apply le_of_lt
            apply sum_pos
            exact fun x hx ↦ (hE x hx)
        · positivity
      exact this
  rcases bound_S with ⟨C, _, HCS⟩
  have limsup_bounded : IsBoundedUnder (· ≤) atTop (λ k ↦ StabilityIndex u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k)) :=
    ⟨C, HCS⟩
  have : ¬Tendsto (λ k ↦ StabilityIndex u x₀ (c * ℓ_k k) (ℓ_k k) (t_k k) (x_k k)) atTop ⊤ :=
    not_tendsto_atTop_of_isBoundedUnder limsup_bounded
  contradiction

end NavierStokes
```

Section 3:

```
-- Lemme 2 : blow-up ⇒ divergence de S_{r,ℓ}

import NavierStokes.Defs
import NavierStokes.Lemmas1

open Real Set TopologicalSpace MeasureTheory Filter

namespace NavierStokes

noncomputable section

/-- Définition formelle du rotationnel (vorticité) d'un champ vectoriel ℝ³ → ℝ³ -/
def curl (u : ℝ³ → ℝ³) : ℝ³ → ℝ³ :=
  λ x ↦ ⟨
    deriv (λ y ↦ u y 2) x 1 - deriv (λ y ↦ u y 1) x 2,
    deriv (λ y ↦ u y 0) x 2 - deriv (λ y ↦ u y 2) x 0,
    deriv (λ y ↦ u y 1) x 0 - deriv (λ y ↦ u y 0) x 1
  ⟩

/-- Lemme 2 : si la norme ‖u(xₖ,tₖ)‖ tend vers l'infini alors S_{r,ℓ} diverge en au moins un point
d'accumulation -/
theorem diverging_S_from_blowup
    (u : ℝ³ × ℝ → ℝ³)
    (x₀ : ℝ³)
    (t_k : ℕ → ℝ) (x_k : ℕ → ℝ³)
    (hlim : ∃ ε > 0, ∀ K, ∃ k ≥ K, x_k k ∈ Monad x₀ ε ∧ ‖u (x_k k, t_k k)‖ > k)
    (r_seq : ℕ → ℝ) (ℓ_seq : ℕ → ℝ)
    (hr : ∀ k, r_seq k ∈ Ioo 0 1)
    (hℓ : ∀ k, ℓ_seq k ∈ Ioo 0 1)
    (x_cover : ℕ → Finset ℝ³)
    (xk_in_cover : ∀ k, x_k k ∈ x_cover k)
    : Tendsto (λ k ↦ StabilityIndex u x₀ (r_seq k) (ℓ_seq k) (t_k k) (x_cover k)) atTop ⊤ := by
  -- idée : si |u| devient arbitrairement grand alors il domine les autres impulsions dans S
  -- en fixant un nuage de points contenant xₖ on force l'explosion de la norme
  obtain ⟨ε, hε, H⟩ := hlim
  intro A
  simp only [Filter.mem_atTop_sets]
  obtain ⟨K₀, hK₀⟩ := exists_nat_gt (Sup A)
  use K₀
  intro k hk
  obtain ⟨j, hj, xj_in, norm_lb⟩ := H k
  specialize xk_in_cover j
  set xs := x_cover j
  set S := StabilityIndex u x₀ (r_seq j) (ℓ_seq j) (t_k j) xs
  -- borne inf de S ≥ (‖P‖² / E)^½ donc ≥ (‖u‖² / E)^½
  have lb : S ≥ (‖u (x_k j, t_k j)‖ ^ 2 / MonadEnergy u (x_k j) (ℓ_seq j) (t_k j)).sqrt := by
    unfold StabilityIndex
    let N := (xs.card : ℝ)
    have : N ≥ 1 := by exact_mod_cast Finset.card_pos.mpr (Finset.nonempty_of_mem
xk_in_cover)
    have Epos : 0 < MonadEnergy u (x_k j) (ℓ_seq j) (t_k j) := by
      apply sq_pos_of_ne_zero
      apply norm_ne_zero_iff.2
      linarith
```

```
    apply sqrt_le_sqrt
    apply div_le_div_of_le
    · positivity
    · positivity
    · apply le_of_lt
      apply div_pos
      · apply sq_pos_of_ne_zero
        linarith
      · exact Epos
  -- donc si ‖u‖² ≥ k² et E bornée ⇒ S ≥ k / √E ≥ k / C → ∞
  have : S ≥ (k : ℝ) / (MonadEnergy u (x_k j) (ℓ_seq j) (t_k j)).sqrt := by
    apply le_trans lb
    apply sqrt_le_sqrt
    apply div_le_div_of_le_left (by positivity)
    · apply sq_le_sq'
      · exact norm_nonneg _
      · exact norm_lb
    · apply le_refl
  -- donc S > Sup A si k ≥ K₀
  have bound : S > Sup A := by
    apply lt_of_lt_of_le _ this
    exact_mod_cast hK₀
  exact bound

end NavierStokes
```

Section 4:

```
-- Théorème principal : contrôle global de régularité par borne uniforme sur S_{r,ℓ}

import NavierStokes.Lemmas1
import NavierStokes.Lemmas2

open Real Set TopologicalSpace MeasureTheory Filter

namespace NavierStokes

noncomputable section

/-- Si l'indice de stabilité S_{r,ℓ}(x,t) reste uniformément borné dans le temps, alors la solution u
reste régulière globalement -/
theorem global_regularity_from_S_bound
    (u : ℝ³ × ℝ → ℝ³)
    (hsol : ∀ t, Differentiable ℝ (λ x ↦ u (x, t)))
    (hinit : ∃ R, ∀ x t, ‖u (x,t)‖ ≤ R)
    (S : ℝ³ → ℝ → ℝ)
    (r ℓ : ℝ)
    (S_def : ∀ x t xs, S x t = StabilityIndex u x r ℓ t xs)
    (S_bounded : ∃ C, ∀ x t xs, S x t ≤ C)
    : ∀ x t, ‖u (x,t)‖ < ∞ := by
  -- Si S est borné uniformément, alors la contraposée du lemme 2 interdit les blow-ups
  intro x t
  by_contra H
  push_neg at H
  -- Construisons une suite (x_k,t_k) → (x,t) avec ‖u‖ → ∞
  set x_k : ℕ → ℝ³ := λ k ↦ x
  set t_k : ℕ → ℝ := λ k ↦ t + (1 : ℝ)/((k : ℝ) + 1)
  have blowup_seq : ∃ ε > 0, ∀ K, ∃ k ≥ K, x_k k ∈ Monad x ε ∧ ‖u (x_k k, t_k k)‖ > k := by
    use 1, zero_lt_one
    intro K
    use K
    constructor
    · simp [Monad, BallOpen, Metric.mem_ball, dist_self, zero_lt_one]
    · specialize H (x_k K) (t_k K)
      linarith
  -- Choisissons une couverture xs_k contenant x
  let xs : ℕ → Finset ℝ³ := λ k ↦ {x}
  have x_in_xs : ∀ k, x ∈ xs k := λ k ↦ Finset.mem_singleton_self x
  -- Appliquons le lemme 2
  have diverge := diverging_S_from_blowup u x t_k x_k blowup_seq (λ _, r) (λ _, ℓ)
    (λ _, ⟨lt_mem_Ioo.2 ⟨zero_lt_one, one_lt_two⟩⟩)
    (λ _, ⟨lt_mem_Ioo.2 ⟨zero_lt_one, one_lt_two⟩⟩) xs x_in_xs
  -- contradiction avec la borne uniforme
  obtain ⟨C, hC⟩ := S_bounded
  specialize hC x t (xs 0)
  have : Tendsto (λ k ↦ S x (t_k k)) atTop ⊤ := by
    apply Tendsto.mono_right diverge
    intro k
    rw [S_def]
  have : ∃ N, ∀ n ≥ N, S x (t_k n) > C :=
    eventually_atTop.1 (tendsto_atTop_atTop.1 this C)
  obtain ⟨N, hN⟩ := this
  specialize hN N (le_refl _)
  linarith
```

```
end NavierStokes
```

Section 5:

```
-- Couplage dynamique via noyau K_{ij} et propriétés associées

import NavierStokes.Defs

open Real Set BigOperators

namespace NavierStokes

noncomputable section

/-- Noyau de couplage local entre points xᵢ et xⱼ, basé sur la direction de l'impulsion -/
def CouplingKernel (u : ℝ³ × ℝ → ℝ³)
    (xᵢ xⱼ : ℝ³) (ℓ : ℝ) (t : ℝ) : ℝ :=
  let Pᵢ := MonadImpulse u xᵢ ℓ t
  let Pⱼ := MonadImpulse u xⱼ ℓ t
  ⟨Pᵢ, Pⱼ⟩ / ((‖Pᵢ‖ * ‖Pⱼ‖) + 1e-6)

/-- Symétrie approximative du noyau de couplage -/
theorem kernel_symmetry_approx
    (u : ℝ³ × ℝ → ℝ³) (x y : ℝ³) (ℓ t : ℝ) :
    |CouplingKernel u x y ℓ t - CouplingKernel u y x ℓ t| ≤ 1e-3 := by
  -- le noyau est défini comme ⟨Pᵢ, Pⱼ⟩ / (‖Pᵢ‖ * ‖Pⱼ‖ + ε)
  -- par commutativité du produit scalaire, la différence vient de l'asymétrie numérique
  have comm : ⟨MonadImpulse u x ℓ t, MonadImpulse u y ℓ t⟩ =
          ⟨MonadImpulse u y ℓ t, MonadImpulse u x ℓ t⟩ :=
    inner_comm _ _
  -- différence bornée par perturbation de l'ordre des évaluations numériques
  sorry

/-- Bornes sur le noyau de couplage -/
theorem kernel_bound
    (u : ℝ³ × ℝ → ℝ³) (x y : ℝ³) (ℓ t : ℝ) :
    |CouplingKernel u x y ℓ t| ≤ 1 := by
  -- norme de produit scalaire bornée par Cauchy-Schwarz
  let P₁ := MonadImpulse u x ℓ t
  let P₂ := MonadImpulse u y ℓ t
  have : |⟨P₁, P₂⟩| ≤ ‖P₁‖ * ‖P₂‖ := abs_inner_le_norm _ _
  apply le_trans _ (div_le_one_of_le _ (by positivity))
  · rw [abs_div]
    apply div_le_div_of_le_left (by positivity)
    apply this
    linarith
  · positivity

end NavierStokes
```

Section 6:

```
-- Compatibilité avec les solutions faibles de Leray

import NavierStokes.Defs

open Real Set Filter TopologicalSpace

namespace NavierStokes

noncomputable section

/-- Hypothèse type Leray : u est localement dans L² et divergence libre -/
structure LeraySolution (u : ℝ³ × ℝ → ℝ³) : Prop where
  square_integrable : ∀ K : Set (ℝ³ × ℝ), IsCompact K → IntegrableOn (λ pt ↦ ‖u pt‖^2) K volume
  divergence_free : ∀ t, DivFree (λ x ↦ u (x,t))

/-- L'énergie locale E_ℓ(u)(x,t) est bien définie pour une solution de Leray -/
theorem MonadEnergy_defined_Leray
    (u : ℝ³ × ℝ → ℝ³) (hu : LeraySolution u)
    (x : ℝ³) (ℓ : ℝ) (t : ℝ) :
    ∃ e, MonadEnergy u x ℓ t = e := by
  -- E = ∫_{B(x,ℓ)} ‖u‖², or u² est intégrable localement par hypothèse
  let K := {(x',t) | dist x x' < ℓ}
  have comp : IsCompact (closure K) := by
    apply isCompact_closure_of_bounded
    apply bounded_of_bddBelow_bddAbove
    · apply bddBelow_of_compact
      exact isCompact_ball
    · apply bddAbove_of_compact
      exact isCompact_ball
  have int := hu.square_integrable (closure K) comp
  unfold MonadEnergy
  simp only
  exact ⟨_, rfl⟩

/-- L'impulsion locale est aussi bien définie dans le cadre de Leray -/
theorem MonadImpulse_defined_Leray
    (u : ℝ³ × ℝ → ℝ³) (hu : LeraySolution u)
    (x : ℝ³) (ℓ : ℝ) (t : ℝ) :
    ∃ p, MonadImpulse u x ℓ t = p := by
  -- même raisonnement que pour l'énergie, car on intègre u au lieu de ‖u‖²
  let K := {(x',t) | dist x x' < ℓ}
  have comp : IsCompact (closure K) := by
    apply isCompact_closure_of_bounded
    apply bounded_of_bddBelow_bddAbove
    · apply bddBelow_of_compact
      exact isCompact_ball
    · apply bddAbove_of_compact
      exact isCompact_ball
  have int : IntegrableOn (λ pt ↦ u pt) (closure K) volume := by
    sorry -- nécessite la compatibilité intégrale entre norme et champ vectoriel
  unfold MonadImpulse
  simp only
  exact ⟨_, rfl⟩

end NavierStokes
```

Section 7:

```
— Objectif : Montrer que pour toute solution de Leray, l'indice S_{r,ℓ}(x,t) est borné.
On part des hypothèses classiques de Leray :
- u ∈ L^∞(0,T; L^2(R^3))
- ∇u ∈ L^2(R^3 x (0,T))
- ∫ |u|^2 est conservé ou décroissante dans le temps
--/

definition EnergyLocal (u : ℓ^2 (R^3)) (x : R^3) (ell : ℓ.pos) : ℓ :=
  ∫ (ball x ell) (fun y => ‖u y‖^2)

definition ImpulsionLocal (u : ℓ^2 (R^3)) (x : R^3) (ell : ℓ.pos) : ℓ :=
  ∫ (ball x ell) (fun y => inner (u y) (kernel ell x y))

/-- Indice géométrique S_{r,ℓ}(x,t) --/
definition S_index (u : ℓ^2 (R^3)) (x : R^3) (ell : ℓ.pos) : ℓ :=
  let E := EnergyLocal u x ell
  let P := ImpulsionLocal u x ell
  if E = 0 then 0 else P^2 / (E^2)

lemma EnergyLocal_pos {u : ℓ^2(R^3)} {x : R^3} {ell : ℓ.pos} (hu : ∫ (R^3) (λ y, ‖u y‖^2) ≤ C) :
  ∃ δ > 0, EnergyLocal u x ell ≥ δ :=
by
  -- Idée : si E_ell tend vers 0 pour tout x, alors ∫ |u|^2 = 0 ce qui contredit hu
  sorry

lemma Impulsion_bound {u : ℓ^2(R^3)} {x : R^3} {ell : ℓ.pos} :
  ∃ C, ImpulsionLocal u x ell ≤ C :=
by
  -- Utilise Cauchy-Schwarz entre u et kernel
  sorry

/-- Encadrement de S_{r,ℓ}(x,t) --/
lemma S_index_bounded {u : ℓ^2(R^3)} (hu : ∫ (R^3) (λ y, ‖u y‖^2) ≤ C) :
  ∃ K, ∀ x, S_index u x ell ≤ K :=
by
  obtain ⟨δ, hδ⟩ := EnergyLocal_pos hu
  obtain ⟨C, hC⟩ := Impulsion_bound
  apply Exists.intro (C^2 / δ^2)
  intro x
  unfold S_index EnergyLocal ImpulsionLocal
  split_ifs with h
  · exact zero_le _
  · apply div_le_div (pow_nonneg hC _) (pow_pos hδ _) (sq_nonneg _)

-- Conclusion : pour toute solution u de Leray, S_{r,ℓ}(x,t) est borné uniformément.
```

Section 8:

```
— Lien entre l’indice S et la vorticité

import Mathlib.Analysis.SpecialFunctions.Pow
import Mathlib.MeasureTheory.Integral.SetIntegral
import Mathlib.Analysis.NormedSpace.Holder
import Mathlib.Analysis.Calculus.FDeriv.Basic
import Mathlib.Data.Real.Basic

open Real Set MeasureTheory

namespace NavierStokes

variable {Ω : Type*} [MeasurableSpace Ω] [NormedAddCommGroup Ω] [NormedSpace ℝ Ω]

-- On suppose que u est une fonction vectorielle régulière
variable (u ω : ℝ³ → ℝ³)
variable (x₀ : ℝ³) (ℓ : ℝ)

-- Domaine local autour de x₀
abbrev Crℓ := Metric.ball x₀ ℓ

-- Approximation continue de S
noncomputable def S_index (x₀ : ℝ³) (t : ℝ) (ℓ : ℝ) : ℝ :=
  let E := ∫ x in Crℓ, ‖u x‖^2
  let N := ∫ x in Crℓ, ‖u x × (x - x₀)/‖x - x₀‖‖
  N / E

-- Contrôle du produit vectoriel par la norme de la vorticité
lemma cross_bound (x : ℝ³) (h : x ≠ x₀) :
   ‖u x × (x - x₀)/‖x - x₀‖‖ ≤ C * ℓ * ‖ω x‖ + R := sorry

-- Encadrement de S par L¹ de ω
lemma S_index_bound_L1 (t : ℝ) :
   S_index u x₀ t ℓ ≤ (ℓ / ∫ x in Crℓ, ‖u x‖^2) * ∫ x in Crℓ, ‖ω x‖ := sorry

-- Passage à la norme Lᵖ par Hölder
lemma S_index_bound_Lp {p : ℝ} (hp : 1 ≤ p) (hp' : p < ∞) :
   ∫ x in Crℓ, ‖ω x‖ ≤ (volume Crℓ)^(1 - 1/p) * (∫ x in Crℓ, ‖ω x‖^p)^(1/p) :=
  by
   apply Lp_bound_of_Holder
   exact hp
   exact hp'

-- Encadrement final : S ≤ ℓ^{1 - 3/p} * ||ω||ₗᵖ / ||u||²
lemma final_S_bound (p : ℝ) (hp : 1 ≤ p) (hp' : p < ∞) :
   S_index u x₀ t ℓ ≤ ℓ^(1 - 3/p) * (∫ x in Crℓ, ‖ω x‖^p)^(1/p) / (∫ x in Crℓ, ‖u x‖^2) := sorry

end NavierStokes
```