Day - 11 : Sequences and Convergence

Touseef Haider

Study Mathematics In Lean(MIL) Section 3.6

```
import Mathlib.Data.Real.Basic

def ConvergesTo (s : ℕ → ℝ) (a : ℝ) :=
  ∀ ϵ > 0, ∃ N, ∀ n ≥ N, |s n - a| < ϵ

-- (s : ℕ → ℝ) is the first input. it is a sequence of real numbers.
-- (a : ℝ) is a real number that the sequence converges to.
-- rest is just convergence definition.

example : (fun x y : ℝ ↦ (x + y) ^ 2) = fun x y : ℝ ↦ x ^ 2 + 2 * x * y + y ^ 2 := by
-- fun x y : ℝ ↦ ... is how Lean writes lambda functions.
  ext
  -- ext is a tactic that stands for ''extensionality''. For functions, it means that to prove two functions are equal, you need to prove they give the same output for every possible input.
  -- ring is a powerful tactic that can automatically prove identities that hold in any ring
  ring
```

```

example (a b : ℝ) : |a| = |a - b + b| := by
  -- congr tactic short for congruence tries to prove equalities by breaking them down. Here, it sees that we are trying to prove |X| = |Y|. It will change the goal to proving X = Y, if it can justify that X=Y implies |X| = |Y|.
  congr
  ring
```

```
example {a : ℝ} (h : 1 < a) : a < a * a := by
  convert (mul_lt_mul_right _).2 h
  -- mul_lt_mul_right is a theorem that eseentially says if x<y and z>0 then x*z < y*z. (_).2 is selecting a specific implication.
  -- We are also telling Lean we want to prove our goal by showing it is equivalent to h, after multiplying both sides by some positive number.
  -- convert tactic that asks Lean to change the goal to something else, provided we can prove the new thing implies the old thing. It changes the goal to 1 * a < a * a, and leaves us with thwo side goals:
  -- 1. Prove that 1 * a = a
  -- 2. Prove that a is positive.
  · rw [one_mul]
    -- rw [one_mul] address the first goal, It rewrites the term 1 * a to just a, because multiplying by 1 does not change the value.
  exact lt_trans zero_lt_one h
  -- This addresses the second goal. It uses the transitivity of the less than relation to show that since 0 < 1 and 1 < a, then 0 < a.
```

Below is the theorem that states that a constant sequence converges to the constant value.

```
theorem convergesTo_const (a : ℝ) : ConvergesTo (fun _ ↦ a) a := by

  intro ϵ ϵpos
  -- We need to prove ∀ ϵ >0 , . . . The intro tactic introduces variables for the universal quantifiers. It introduces ϵ(a real number) and ϵpos(a proof that ϵ > 0). Our new goal will be ∃ N, ∀ n ≥ N, |a - a| < ϵ.
  use 0
  -- We need to prove ∃ N, ... The use tactic provides a witness for the existenial quantifier. Here, we use 0 as the witness for N. Our new goal is ∀ n ≥ 0, |a - a| < ϵ.
```

```
  intro n nge
  -- We need to prove ∀ n ≥  0, ... We introduce n (a natural number) and nge (a proof that n ≥ 0). Our new goal is |a - a| < ε.
  rw [sub_self, abs_zero]
  -- sub_self rewrites a-a as 0. The goal becomes |0| < ε.
  -- abs_zero rewrites |0| as 0. The goal becomes 0 < ε.

  apply εpos
  -- apply tactic tries to prove the goal by applying a theorem or lemma. Here, it applies εpos, which is a proof that ε > 0. The goal is now proved.
```

Below is the theorem that states that the sum of two convergent sequences converges to the sum of their limits.

```
theorem convergesTo_add {s t : ℕ → ℝ} {a b : ℝ}
      (cs : ConvergesTo s a) (ct : ConvergesTo t b) :
    ConvergesTo (fun n ↦ s n + t n) (a + b) := by
  intro ε εpos
  -- Introduce ε and εpos, which is a proof that ε > 0. The goal is now to find an N such that for all n ≥ N, |s n + t n - (a + b)| < ε.
  dsimp -- this tactic unfolds the definition of ConvergesTo, making the goal clearer.
  have ε2pos : 0 < ε / 2 := by linarith -- This is a key step in many convergence proofs. We want to use the fact that s and t converge. We will need them to be closer than ε/2 to their limits.
  -- ε2pos is a proof that ε / 2 > 0. We use linarith to derive this from εpos.
  rcases cs (ε / 2) ε2pos with ⟨ Ns, hs ⟩
  -- This is where we use the hypothesis cs. Since s converges, the definition ConvergesTo s a holds for any positive number including ε / 2 (using ε2pos).
  -- So cs (ε / 2) ε2pos gives us a natural number Ns and a proof hs that for all n ≥ Ns, |s n - a| < ε / 2.
  -- rcases with ⟨ Ns, hs ⟩ means we are unpacking the existential and universal quantifier. It extract the N (which it names Ns) and the proof ∀ n ≥ Ns, |s n - a| < ε / 2 (which it names hs).

  rcases ct (ε / 2) ε2pos with ⟨ Nt, ht ⟩
  -- This is similar to the previous step.
  use max Ns Nt
  -- Now we need to provide our N for the sum sequence. We choose the maximum of Ns and Nt. This ensures that both conditions for s and t are satisfied for n greater than or equal to this N.
  intro n nge
  -- We introduce n and nge that n ≥ max Ns Nt. The goal is now to show that |s n + t n - (a + b)| < ε.
  have hns : n ≥ Ns := le_of_max_le_left nge
  -- We need to show that n ≥ Ns. Since n ≥ max Ns Nt, and the maximum is always greater than or equal to its left component, we can conclude n ≥ Ns. le_of_max_le_left is the theorem that says this, and nge is our proof that n ≥ max Ns Nt.
  have hnt : n ≥ Nt := le_of_max_le_right nge
  calc
    |s n + t n - (a + b)| = |(s n - a) + (t n - b)| := by
      congr; ring -- We rearrange the terms inside the absolute value. congr simplifies it to proving the insdie is equal and ring prove that.
    _ ≤ |s n - a| + |t n - b| := abs_add _ _ -- We use the triangle inequality (abs_add) which states |x + y| ≤ |x| + |y| for any real numbers x and y.
    _ < ε / 2 + ε / 2 := add_lt_add (hs n hns) (ht n hnt) -- We know from hs that if n ≥ Ns (which hns proves), then |s n - a| < ε / 2. So hs n hns is the proof of |s n - a| < ε / 2. Similarly, ht n hnt gives us |t n - b| < ε / 2.
    -- add_lt_add is a theorem that states if x < y and z < w, then x + z < y + w. We use it to combine the two inequalities.
    _ = ε := by ring -- Finally, we use the ring tactic to simplify ε / 2 + ε / 2 to ε. This completes the proof that the sum of the sequences converges to the sum of their limits.
```

We are proving that if s converges to a , then (c * s) converges to (c * a)

```
theorem convergesTo_mul_const {s : ℕ → ℝ} {a : ℝ} (c : ℝ) (cs : ConvergesTo s a) :
    ConvergesTo (fun n ↦ c * s n) (c * a) := by
  -- First, let's handle the easy case where c is 0.
  by_cases h : c = 0
  -- This is the block for the c=0 case.
```

```
    -- convert changes the goal to proving that two things are equal.
    -- Here, it changes the goal to proving ‘convergesTo_const 0’ is the same as our original goal.
  · convert convergesTo_const 0
    -- The first goal from ’convert’ is to show ’(fun n ↦ c * s n) = (fun n ↦ 0)
    · rw [h] -- replace c with 0
      ring -- The ring tatic can solve equations like 0 * s n = 0
    -- The second goal is to show c * a = 0.
    rw [h] -- replace c with 0
    ring -- ring solve 0 * a = 0

  -- This is the blok for c ≠ 0 case. h is now hypothesis c ≠ 0.
  have acpos : 0 < |c| := abs_pos.mpr h
  -- our goal is to show ConvrgesTo (fun n ↦ c * s n) (c * a)

  -- Start the ϵ - N proof
  intro ϵ ϵ_pos

  -- OUr hypothesis cs gives us an N for any positive epsilon.
  -- We will use it with ϵ’ = ϵ / |c|
  -- This is a valid positive epsilon because ϵ > 0 and |c| > 0.
  have ϵ’_pos : ϵ /|c|  > 0 := div_pos ϵ_pos acpos

  -- Get the N from our hypothesis cs.
  let ⟨ N, hN ⟩ := cs (ϵ /|c|) ϵ’_pos

  -- Now we prove this N as our witness for the final goal.

  use N

  -- We now have to prove that for any n ≥ N , the inequality holds.
  intro n hn

  -- The goal is now | (fun n ↦ c * s n ) n - c* a| < ϵ
  -- Let’s simplify the expression
  rw [← mul_sub, abs_mul]

  -- The goal is now |c|*|s n - a| < ϵ
  -- We can use the inequality from our hypothesis ’hN’
  -- hN says that for n ≥ N, we have |s n - a| < ϵ/|c|
  -- So we can replace |s n - a| with something bigger to prove our goal
  calc
    |c| * |s n - a| < |c| * ( ϵ/|c|) := by
      -- this step is true because |s n - a| < ϵ /|c|
      apply mul_lt_mul_of_pos_left
      · exact hN n hn
      · exact acpos
    _ = ϵ := by
      -- This step is just algebra
      rw [mul_div_cancel₀ _ (ne_of_gt acpos)]
```