UNTITLED LEAN THESIS

A Thesis Submitted to the Faculty of

Georgetown College

In Partial Fulfillment of the Requirements for the

Honors Program

By

Logan Johnson

Georgetown, Kentucky

May 2024

Abstract

UNTITLED LEAN THESIS

Logan C. Johnson

idk dr burch?? maybe dr white???

Here is the text of your abstract. It goes on and on and on. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. The rest of this paragraph is a filler. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. It goes on like this for about 150 words, so it should all fit on this page. Note that the Abstract comes before the title page and has no page number. If your abstract is more than 250 words, consider shortening it.

APPROVED BY THE DIRECTOR OF HONORS THESES:

____

Dr. Homer White, Department of Mathematics

APPROVED BY THE HONORS PROGRAM:

____

Dr. Barbara Burch, Director

DATE:____________________

Table of contents

Preface

1 Real Analysis

2 Functional Programming

3 Lean as a Theorem Prover

Differences From Paragraph Style Proofs

Despite the incredible power that lean could provide in the verification of mathematical proofs, this does pose some difficulties, namely the ease with which the aforementioned proofs can be written up. Typically, proofs are simply written up in a paragraph style, where the steps being taken and the theorems being applied are laid out in plain terms so that it can be easily understood by fellow mathematicians. There are often times when mathematicians will take things for granted or skip over steps that they think the reader will either already know to be fact or can easily reason out for themselves when writing out typical proofs. This lax approach for conveying information simply does not work when trying to communicate with technology, and a much more specific and methodical approach must be adopted in order to take advantage of the logical verification benefits. Thankfully, lean has a community working to create libraries of previously proven theorems that can be applied to speed up the writing and verification of future proofs. This thankfully means that all proofs do not need to be taken all the way back to basic axioms: Users can save time by avoiding proving adjacent theorems and instead focus only on the immediately relevant steps of their proof.

For each of the following proofs, I will first provide a typical "paragraph style" version of the proof, so the differences between the two can easily be compared.

Inequality Addition

Paragraph Style Proof

Theorem. *If $a < b$ and $c \leq d$, prove that $a + c < b + d$*

There are multiple ways to approach this in a paragraph style proof, so I will attempt to have this proof follow along the same lines as the lean proof.

Proof. There are two possible cases: either $c = d$ or $c < d$. We will first consider the case where $c = d$. We know $a < b$, so it would also be true that $a + c < b + c$. Then because $c = d$, $a + c < b + d$. Now consider the case where $c < d$. We know $a < b$, so $a+c < b+c$ and $b+c < b+d$ because $c < d$. Thus by transitivity of inequalities, we could say $a + c < b + d$ □

Lean Proof

Seting up the problem

Here I put the theorem we want to prove into lean and we can see the resulting infoview panel. I name our two assumptions `h1` and `h2`, for hypotheses one and two. After a colon I then write out the thing I am trying to prove with those hypotheses and use `by` to put lean into tactic mode.

It can now be seen that the infoview panel lists out both of our hypotheses as well as the goal we are working towards at the bottom. This panel will continue to change as more code is added to the lean file.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
⊢ a + c < b + d
```

Step 1

Here I lay out the two possible cases of our second hypothesis which allows me to strengthen the information that we know. We see this strengthened hypothesis reflected in `h3` in the infoview.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
h3: c = d
⊢ a + c < b + d
```

Step 2

Here I used hypothesis 3 to rewrite he c in our final goal as a d. This change is reflected in the infoview for this step.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
h3: c = d
⊢ a + d < b + d
```

Step 3

In this step I applied a theorem already in the Mathlib library for lean. The `add_lt_add_right` theorem simply states that if you have a `b < c`, then `b + a < c + a` which is exactly what we need to prove the goal for the first case. As the first case has been completed, the infoview then switches to the second case which is reflected in the new `h3` and reset goal.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]
  apply add_lt_add_right h1

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
h3: ¬c = d
⊢ a + c < b + d
```

Step 4

In order to better work with our new hypothesis, I use a tactic which pushes the negation symbol further into the thing it is negating. This results in a hypothesis which can actually be applied later on.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]
  apply add_lt_add_right h1
  push_neg at h3

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
h3: c ≠ d
⊢ a + c < b + d
```

Step 5

Here I am laying out a new hypothesis which will be useful later in the proof. This hypothesis seems like an obvious conclusion based on hypotheses two and three, but we must still lay it out simply for lean if we want to actually use it. The infoview panel always displays the most current goal, which is why it is displaying the goal for `h4` rather than the main goal.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]
  apply add_lt_add_right h1
  push_neg at h3
  have h4 : c < d := by

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
abcd: ℝ
h1: a < b
h2: c ≤ d
h3: c ≠ d
⊢ c < d
```

Step 6

Here I apply another theorem already in lean which takes the information `h3` and `h2` gives us and shows our current goal. Writing out `h4` like this is technically optional, as lean allows you to evaluate tactics within arguments for other tactics. Despite this, I personally find it more convenient and clear to write out extra hypotheses like this rather than just giving the body of the argument when necessary. Now that our new hypothesis has been proven, the infoview displays that we have no goals until we get back into our main theorem.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]
  apply add_lt_add_right h1
  push_neg at h3
  have h4 : c < d := by
    apply Ne.lt_of_le h3 h2

  done
```

Tactic State in Infoview

```
No goals
```

Step 7

I now use the calc tactic to work through the rest of the theorem. This tactic is quite useful as it allows us to chain together multiple equalities or inequalities while still giving proofs for each step. This is essentially a shortcut of writing out individual hypotheses and then using the rewrite tactic to get our desired goal.

In this case, I only need to do two steps of chaining inequalities, where I use transitivity to show that the starting value is less than the final value. It essentially follows the same path as the paragraph style proof, where the tactics `add_lt_add_right` and `add_lt_add_left` justify the steps taken.

Lean File

```
example (a b c d : ℝ) (h1: a < b)
    (h2 : c ≤ d) : a + c < b + d := by
  by_cases h3 : c = d
  rw [h3]
  apply add_lt_add_right h1
  push_neg at h3
  have h4 : c < d := by
    apply Ne.lt_of_le h3 h2
  exact calc
    a + c < b + c := add_lt_add_right h1 c
    _ < b + d := add_lt_add_left h4 b
  done
```

Tactic State in Infoview

```
No goals
```

a is Less Than or Equal to b

Paragraph Style Proof

Theorem. *Suppose that $a, b \in \mathbb{R}$ and for every $\varepsilon > 0$, we have $a \leq b + \varepsilon$. Show that $a \leq b$.*

Proof. Assume for the sake of contradiction that a is not less than or equal to b. Then it would be true that $a > b$. Now consider the case where $\varepsilon = \frac{a-b}{2}$. Then since $a > b$, epsilon is positive and by our assumption then

$a \leq b + \varepsilon$. Then

$$\begin{aligned} a &\leq b + \varepsilon \\ &= b + \frac{a-b}{2} \\ &= b + \frac{a}{2} - \frac{b}{2} \\ &= \frac{a}{2} + \frac{b}{2}. \end{aligned}$$

So now,

$$\begin{aligned} a &\leq \frac{a}{2} + \frac{b}{2} \\ a - \frac{a}{2} &\leq \frac{b}{2} \\ \frac{a}{2} &\leq \frac{b}{2} \\ a &\leq b. \end{aligned}$$

But now we have that $a \leq b$ and $a > b$, a contradiction! $\square$

Lean Proof

Setting up the problem

I again set up the proof with our one hypothesis and the goal we want to prove. These are then seen listed in the infoview on the right.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
⊢ a ≤ b
```

Step 1

The `by_contra` tactic allows me to complete this problem using proof by contradiction. This tactic automatically creates a hypothesis containing the negation of the final goal, I named it `h2`, and changes the final goal to `False` meaning that it needs a contradiction.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: ¬a ≤ b
⊢ False
```

Step 2

Here I use the `push_neg` tactic similarly to the previous example to het a usable version of `h2` as well as pick a specific epsilon for which we will find a contradiction. This new epsilon will now show up in the infoview in the side and can be used in our problem.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
⊢ False
```

Step 3

Here I lay out a hypothesis that we will later be able to apply to `h1`. Saying that epsilon was positive in the paragraph style proof fairly simpler, where we only really need to justify that $a - b$ is positive. In lean however, it requires a bit more effort and as such I put in its own hypothesis.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by

    done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
⊢ ε > 0
```

Step 4

Anyone reading a paragraph style proof such as ours would know that dividing a number by two does not impact whether the resulting number if positive or negative, but it still needs to be justified to lean. As such, I use the `half_pos` theorem with the `refine` tactic to change the goal to what is currently shown in the infoview. The `refine` tactic is useful because it tries to apply the arguments it is given to the final goal and change the goal to whatever is needed to meet the hypotheses in the arguments. In this case, `half_pos` claims that if you have some $a > 0$. then $\frac{a}{2} > 0$. The `refine` tactic then applies the result of that theorem and leaves us to show that $a > 0$, and lean is smart enough to figure out that we actually need to show $a - b > 0$.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h

      done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
⊢ 0 < a - b
```

Step 5

The theorem I uses our second hypothesis to show that $a - b > 0$, which finishes the proof for our third hypothesis and the goal switches back to finding a contradiction.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
h3: ε > 0
⊢ False
```

Step 6

I now try to lay out the fourth and final hypothesis which will be used to find a contradiction with `h2`. This is another example of something being quickly explained in the paragraph style proof, but being more cumbersome to justify within lean.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done
  have h4 : a ≤ b + ε := by

    done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
h3: ε > 0
⊢ a ≤ b + ε
```

Step 7

I first apply `h1` which works has a similar effect as using the refine tactic earler: it applies to result of an if then statement and changes our goal to the if.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done
  have h4 : a ≤ b + ε := by
    apply h1

    done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
h3: ε > 0
⊢ ε > 0
```

Step 8

Now that our goal has been properly modified, `h3` is the only other thing necessary to justify this hypothesis.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done
  have h4 : a ≤ b + ε := by
    apply h1
    apply h3
    done

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
h3: ε > 0
h4: a ≤ b + ε
⊢ False
```

Step 9

The `dsimp` tactic will do its best to automatically simplify anything it is given, in this case it substitutes our specific epsilon value in for the arbitrary epsilon. This will now allow us to use `h4` to find our contradiction.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done
  have h4 : a ≤ b + ε := by
    apply h1
    apply h3
    done
  dsimp at h4

  done
```

Tactic State in Infoview

```
R: Type u_1
inst†: Ring R
ab: ℝ
h1: ∀ (ε : ℝ), ε > 0 →
  a ≤ b + ε
h2: b < a
ε: ℝ := (a - b) / 2
h3: ε > 0
h4: a ≤ b + (a - b) / 2
⊢ False
```

Step 10

The `linarith` tactic is quite powerful as it will attempt to simplify the goal as well as hypotheses and then look for a contradiction amongst the known hypotheses. This is one example where lean actually requires quite a bit less explanation than a typical proof. The majority of my paragraph style proof above was spent simplifying and manipulating `h4` and `h2`, whereas in lean I need to specify none of that! It is quite impressive that lean is already able to do so much simplification and even find contradictions with no user input. This ability will likely only increase in power in the future, and some developments have even occured during the planning and writing of this thesis that make other simplification tactics substantially more powerful.

Lean File

```
example (a b : ℝ) (h1 : ∀ ε : ℝ,
    ε > 0 → a ≤ b + ε) :
    a ≤ b := by
  by_contra h2
  push_neg at h2
  let ε := (a - b) / 2
  have h3 : ε > 0 := by
    refine half_pos ?h
    exact Iff.mpr sub_pos h2
    done
  have h4 : a ≤ b + ε := by
    apply h1
    apply h3
    done
  dsimp at h4
  linarith
  done
```

Tactic State in Infoview

```
No goals
```

Absolute Convergence

Paragraph Style Proof

Theorem. *Prove that* $\lim(x_n) = 0$ *if and only if* $\lim(|x_n|) = 0$.

Proof. ($\Longrightarrow$) First assume that $\lim(x_n) = 0$. Then for all $\varepsilon > 0$ we know there exists a $k_n \in \mathbb{N}$ such that for all nautral numbers $n > k_n$, $|x_n - 0| < \varepsilon$. Thus $|x_n| < \varepsilon$ and also $||x_n| - 0| < \varepsilon$, so $\lim(|x_n|) = 0$.

($\Longleftarrow$) Now assume that $\lim(|x_n|) = 0$. Then for all $\varepsilon > 0$ we know there exists a $k_n \in \mathbb{N}$ such that for all nautral numbers $n > k_n$, $||x_n| - 0| < \varepsilon$. But $||x_n| - 0| = ||x_n|| = |x_n| = |x_n - 0|$. So $|x_n - 0| < \varepsilon$ and $\lim(x_n) = 0$. $\square$

Lean Proof

Setting up the problem

Lean does not include a built in epsilon definition of a limit for sequences, so it is first necessary to define a limit in lean. I use the following definition:

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

From this point we can set up our problem as normal.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
⊢ ConvergesTo s1 0
  ↔ ConvergesTo |s1| 0
```

Step 1

The first thing I ask lean to do is rewrite the definition of convergence that I defined earlier when it is used in our goal. This will allow us to actually use and work towards the information in both instances of `ConvergesTo` in the problem. The fully expanded definition is shown in the infoview panel.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
⊢ (∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |s1 n - 0| < ε) ↔
    ∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |abs s1 n - 0| < ε
```

Step 2

Here I set up a hypothesis which will later be used to modify both sides of the if and only if statement into something that is equal to the other.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by

    done

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
x: ℕ
⊢ |s1 x| = |abs s1 x|
```

Step 3

In this instance lean essentially already knows that our goal is true, and only need to be told to simplify using the definition of absolute value in order to verify this. While it is impressive that lean requires little guidance, seeing some of the other things lean is capable of leaves me a bit underwhelmed that lean requires any input here. Because lean is still being developed there may come a time where simple statements like this are automatically verified without any user input.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
⊢ (∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |s1 n - 0| < ε) ↔
    ∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |abs s1 n - 0| < ε
```

Step 4

Here the `Iff.intro` tactic splits up the if and only if statement in the goal and allows us to prove each direction individually, as is often done in a paragraph style proof.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards

  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
⊢ (∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |s1 n - 0| < ε) →
    ∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |abs s1 n - 0| < ε
```

Step 5

The `intro` tactic applied here allows me to assume the if part of an if then statement and automatically names it with the hypothesis name I give it.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1

  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
h1: ∀ (ε : ℝ), ε > 0 →
  ∃ N, ∀ (n : ℕ),
  n ≥ N → |s1 n - 0| < ε
⊢ ∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |abs s1 n - 0| < ε
```

Step 6

With the `simp` tactic, lean attempts to simplify the current goal. In this case, the $||s1_n| - 0|$ is simplified to $||s1_n||$. This is now where our `h3` hypothesis can be applied, but I will first attempt to simplify `h1`.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp

  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
h1: ∀ (ε : ℝ), ε > 0 →
  ∃ N, ∀ (n : ℕ),
  n ≥ N → |s1 n - 0| < ε
⊢ ∀ (ε : ℝ), 0 < ε →
    ∃ N, ∀ (n : ℕ),
    N ≤ n →
    |abs s1 n| < ε
```

Step 7

The `simp` tactic has the same effect as in the previous step, but this time it is working on `h1` rather than the end goal. By default `simp` will attempt to work on the goal but if asked to it will attempt to simplify hypotheses as well.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp
    simp at h1

  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
h1: ∀ (ε : ℝ), 0 < ε →
  ∃ N, ∀ (n : ℕ),
  N ≤ n → |s1 n| < ε
⊢ ∀ (ε : ℝ), 0 < ε →
    ∃ N, ∀ (n : ℕ),
    N ≤ n →
    |abs s1 n| < ε
```

Step 8

We can now use the reverse direction of `h3` to simplify our goal further. Notice that the leftwards facing arrow is necessary, as lean typically tries to apply equalities from left to right. This means if the left side of the equality does not match what lean is attempting to replace, lean will not be able to rewrite in other terms.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp
    simp at h1
    simp [← h3]

  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
h1: ∀ (ε : ℝ), 0 < ε →
  ∃ N, ∀ (n : ℕ),
  N ≤ n → |s1 n| < ε
⊢ ∀ (ε : ℝ), 0 < ε →
    ∃ N, ∀ (n : ℕ),
    N ≤ n →
    |s1 n| < ε
```

Step 9

The simplification done over the last few steps has modified both `h1` and our goal to be the same thing. Since we are assuming `h1` to be true, this allows us to apply that hypothesis and complete the first direction of out goal. Upon completion of the first goal, lean automatically begins displaying the second goal, which can be solved quite similarly to the first.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp
    simp at h1
    simp [← h3]
    apply h1
  · --Reverse

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
⊢ (∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |abs s1 n - 0| < ε) →
    ∀ (ε : ℝ), ε > 0 →
    ∃ N, ∀ (n : ℕ),
    n ≥ N →
    |s1 n - 0| < ε
```

Step 10

With this direction I try to simplify in the same ways as before, but instead of using the leftwards direction of the euality in `h3`, I use the rightwards direction. This means that arrow does not need to be included and once again he have `h1` equal to our current goal.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp
    simp at h1
    simp [← h3]
    apply h1
  · --Reverse
    intro h1
    simp
    simp at h1
    simp_rw [h3]

  done
```

Tactic State in Infoview

```
s1: ℕ → ℝ
h3: ∀ (x : ℕ), |s1 x| =
  |abs s1 x|
h1: ∀ (ε : ℝ), 0 < ε →
  ∃ N, ∀ (n : ℕ),
  N ≤ n → |abs s1 n| < ε
⊢ ∀ (ε : ℝ), 0 < ε →
    ∃ N, ∀ (n : ℕ),
    N ≤ n →
    |abs s1 n| < ε
```

Step 11

With a hypothesis equal to our goal, we are able to apply the hypothesis and prove the other direction of the if and only if statement, completing the proof.

Lean File

```
example (s1 : ℕ → ℝ) :
    ConvergesTo s1 (0 : ℝ) ↔
    ConvergesTo (abs s1) (0 : ℝ)
    := by
  rw [ConvergesTo]
  rw [ConvergesTo]
  have h3 (x : ℕ) : |s1 x| =
      |abs s1 x| := by
    simp [abs]
    done
  apply Iff.intro
  · --Forwards
    intro h1
    simp
    simp at h1
    simp [← h3]
    apply h1
  · --Reverse
    intro h1
    simp
    simp at h1
    simp_rw [h3]
    apply h1
  done
```

Tactic State in Infoview

```
No goals
```

Convergence of a Specific Sequence

The following is an example of one situation where lean is somewhat lacking in comparison to a paragraph style proof. The paragraph style proof is able to quickly and easily prove the desired end goal, but lean has to work around a lot of the simple rewriting we would do in a normal proof. In this attempt to prove the convergence of a specific sequence, there were many issues with simplification involving arbitrary variables and the change from natural numbers to real numbers. These sorts of things can be easily explained in a paragraph style proof, but required significant work to prove in lean.

I mentioned earlier that lean's ability to simplify and make connections without user input was advancing quickly, and I encountered this when working on this problem. I originally had great difficulty getting lean to accept that $2 = \frac{2(n+1)}{n+1}$, which is something which can easily be explained in a typical proof, but lean has recently strengthened a tactic that renders much of my work here unnecessary. The `field_simp` tactic tries to simplify the current goal using what is known about all fields, and since we are working with the real numbers we are able to take advantage of this. I was not able to use this tactic since it was changed while I was working on the project, but seeing how quickly lean is progressing is very promising.

Lean internally defines limits using filters and topology rather than the real analysis approach of epsilons, so the approach I was taking here is not the optimal approach for theorems involving limits in lean. While this high level definition of a limit is very useful for the people who know how to use it, it makes lean more difficult to use for those who have not yet studied topology. Definitions such as this start to portray that lean is not really something meant to be used for lower level mathematics, but rather complex and high level proofs.

Paragraph Style Proof

Theorem. *Prove that* $\lim(\frac{2n}{n+1}) = 2$.

Proof. Let $\varepsilon > 0$ and choose $k > \frac{1}{\varepsilon} - 1$ where $k \in \mathbb{N}$ by the Archimedean

Property. Then for $n > k$:

$$\begin{aligned}\left|\frac{2n}{n+1} - 2\right| &= \left|\frac{2n}{n+1} - \frac{2(n+1)}{n+1}\right| \\ &= \left|\frac{-1}{n+1}\right| \\ &= \frac{1}{n+1} \\ &< \frac{1}{k+1} \\ &< \frac{1}{\frac{1}{\varepsilon} - 1 + 1} = \varepsilon.\end{aligned}$$

Thus we have that $\lim(\frac{2n}{n+1}) = 2$. $\square$

Lean Proof

```
example : ConvergesTo (fun (n : ℕ) ↦
    ((2 * n) / (n + 1))) 2 := by
  intro ε
  intro h1
  obtain ⟨k, h13⟩ :=
    exists_nat_gt (2 / ε - 1) --Archimedean Property
  use k
  intro n
  intro h2
  dsimp
  have h3 : (2 : ℝ) = 2 * ((n + 1) / (n + 1)) := by
    have h4 : ((n + 1) / (n + 1)) =
        (n + 1) * ((n + 1) : ℝ)⁻¹ := by
      rfl
      done
    rw [h4]
```

```
  have h5 : (n + 1) * ((n + 1) : ℝ)⁻¹ = 1 := by
    rw [mul_inv_cancel]
    exact Nat.cast_add_one_ne_zero n
    done
  rw [h5]
  exact Eq.symm (mul_one 2)
  done
nth_rewrite 2 [h3]
have h6 : 2 * ((↑n + 1) : ℝ) / (↑n + 1) =
    ((2 * n) + 2) / (n + 1) := by
  rw [Distribute n]
  done
have h7 : 2 * (((↑n + 1) : ℝ) / (↑n + 1)) =
    2 * (↑n + 1) / (↑n + 1) := by
  rw [← mul_div_assoc 2 ((n + 1) : ℝ) ((n + 1) : ℝ)]
  done
rw [h7]
rw [h6]
rw [div_sub_div_same (2 * n : ℝ) (2 * n + 2) (n + 1)]
rw [sub_add_cancel']
rw [abs_div]
simp
have h8 : |(↑n + 1 : ℝ)| = ↑n + 1 := by
  simp
  apply LT.lt.le (Nat.cast_add_one_pos ↑n)
  done
rw [h8]
have h9 : (2 : ℝ) / (↑n + 1) ≤ 2 / (k + 1) := by
  apply div_le_div_of_le_left
  · --case 1
    linarith
    done
  · --case 2
```

```
      exact Nat.cast_add_one_pos k
      done
    · --case 3
      convert add_le_add_right h2 1
      apply Iff.intro
      · --subcase 1
        exact fun a => Nat.add_le_add_right h2 1
        done
      · --subcase 2
        intro h14
        apply add_le_add_right
        exact Iff.mpr Nat.cast_le h2
        done
      done
    done
  have h10 : 2 / (k + 1) < 2 / (2 / ε - 1 + 1) := by
    apply div_lt_div_of_lt_left
    · --case 1
      linarith
      done
    · --case 2
      simp
      apply div_pos
      linarith
      apply h1
      done
    · --case 3
      convert add_le_add_right h2 1
      apply Iff.intro
      · --subcase 1
        intro h11
        exact Nat.add_le_add_right h2 1
        done
```

```
        · --subcase 2
          intro h11
          have h12 : 2 / ε - 1 < (k : ℝ) := by
            simp only []
            apply h13
            done
          exact add_lt_add_right h12 1
          done
        done
      done
    calc
      2 / (↑n + 1) ≤ (2 : ℝ) / (k + 1) := by
        apply h9
        done
      _ < (2 : ℝ) / (2 / ε - 1 + 1) := by
        apply h10
        done
      _ = ε := by
        ring_nf
        apply inv_inv
        done
    done
```

4 Conclusions

Ultimately lean is an incredibly powerful tool which does provide the valuable proof verification benefits which make learning the language worthwhile.

5 Works Cited

This work had been formatted and styled from the book *How To Prove It With Lean*, written by Daniel J. Velleman. *How To Prove It With Lean* contains short excerpts from *How To Prove It: A Structured Approach, 3rd Edition*, by Daniel J. Velleman and published by Cambridge University Press.

6 Additional space

Extra chapter to write more things if needed!!