# Formalising Mathematics - Coursework 1

# Intermediate Value Theorem

Szymon Kubica

February 2, 2023

## Introduction

In the first part of the coursework I've decided to formalise the intermediate value theorem which was covered as a part of the course MATH40002: Analysis I in the first year of my course. My choice of this theorem was motivated by the fact that the next two parts of the coursework need to cover concepts that I've learned in my second and third year respectively. Because of this, and the fact that as a JMC student I only took one other maths module this term (which is MATH60029: Functional Analysis), I needed to start building up my knowledge of formalising proofs in mathematical analysis.

This report documents the process of formalising IVT using the Lean programming language. It is a functional language which can also be used as an interactive theorem prover. I used Lean together with the mathlib library which contains many fundamental theorems and identities that are useful to build more complex proofs with.

In the following I state the intermediate value theorem, prove it and explain my approach to formalising it. I document the difficulties that I encounered while learning how to formalise arguments efficiently and focus on the surprising observations that I made during the process. I also introduce an overview of adding new notation to the proof to make it easier to follow.

## Proof of the Intermediate Value Theorem

Before we go over the process of formalising the theorem, let us start by considering its proof.

**Theorem** (Intermediate Value Theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be a continuous function, then for all $c$ between $f(a)$ and $f(b)$, there exists $x \in [a, b]$ satisfying $f(x) = c$.*

*Proof.* Let us first assume without loss of generality that $f(a) < c < f(b)$. We can do this because if we consider the other possible cases, we get the following:

- if either $f(a) = c$ or $f(b) = c$ we take $a$ or $b$ respectively and the claim follows;
- if $f(a) = f(b)$ then c = f(a) and we are done;
- if instead $f(a) < f(b)$, we define $g(x) = -f(x)$, and thus $g(a) < g(b)$. Now we are back in the first case with $g(a) < -c < g(b)$, and so we need to find an $x$ such that $g(x) = -c$ which in turn implies that $f(x) = c$ and the claim follows.

Note that the case analysis above will be important in the formal proof in Lean. It allowed me to prove the theorem just for the case $f(a) < c < f(b)$ as a separate lemma and then reuse it in the main body of the proof. It was also surprisingly not trivial which I will document in the later part of the report.

Now we can define the following set:

$$S = \{y \in [a, b] \mid f(y) < c\}.$$

We'll show that it has a supremum $x$ and that it satisfies $f(x) = c$. First, observe that $S$ is nonempty, as $a \in S$, and it is also bounded above by $b$. Therefore, we may deduce that $S$ has a supremum. Moreover, if we let $x = \sup S$, it satisfies $a \le x \le b$.

The aim now is to show that $f(x) \ge c$ and $f(x) \le c$ and so our goal will follow from antisymmetry.

To prove that $f(x) \ge c$ let us argue by contradiction and assume $f(x) < c$. In in this case, we'll use continuity at $x$ with $\epsilon = c - f(x)$ which satisfies $\epsilon > 0$. Observe that now if $f(x) < c$ and $c < f(b)$, we deduce $x \neq b$, thus we get $x < b$. Using continuity at $x$, we obtain:

$$\exists \delta > 0 \text{ such that } \forall y \in [a, b] \text{ we have } |x - y| < \delta \implies |f(y) - f(x)| < \epsilon.$$

By considering the inequality $|f(y) - f(x)| < \epsilon$ above we can deduce that:

$$\forall y \in (x - \delta, x + \delta) \cap [a, b] \text{ we have } f(y) < f(x) + \epsilon = c.$$

Therefore, by definition of $S$, we have that all $y$ above belong to $S$. If we now let $y = x + \frac{1}{2}\min(\delta, b - x)$, we have found a $y \in S$ which is greater than $x$. However, $x$ was supposed to be an upper bound for $S$, so we have a contradiction.

For the second case $f(x) \le c$, we also argue by contradiction. Similar as before, we assume that $f(x) > c$ and deduce that $x \neq a$. We let $\epsilon = f(x) - c > 0$ and by continuity of $f$ at $x$, we obtain a $\delta > 0$ such that:

$$\forall y \in (x - \delta, x + \delta) \cap [a, b] \text{ we have } |f(y) - f(x)| < \epsilon. \tag{1}$$

Now, if for some $y$ we have $|f(y) - f(x)| < \epsilon$, then in particular $f(x) - \epsilon < f(y)$ and by our definition of $\epsilon$, we have $f(x) - \epsilon = c$. We get $c < f(y)$, and so by definition of $S$ none of such $y$ belong to $S$. By letting:

$$m = \max\left(x - \frac{\delta}{2}, a\right),$$

we can observe that $m < x$ and moreover $(m, x) \subset (x - \delta, x + \delta) \cap [a, b]$. Thus, if we apply the property (1) above, we may deduce that for all $y \in (m, x)$ we have $y \notin S$. Therefore, $m$ is an upper bound for $S$, however it is less than $x$, which contradicts with $x$ being the least upper bound.

Hence we deduce that both $f(x) \geq c$ and $f(x) \leq c$ are satisfied, thus, by antisymmetry, we have $f(x) = c$ which we had to show. ■

Let us now focus on my approach to formalising the IVT and consider the surprising observations that I made in the process.

## The Process of Formalising

In order to formalise the theorem I started by formulating a hand-written proof of the theorem using the course notes. After I familiarised myself with the argument, I tried to directly translate it into the Lean code without thinking about the overall structure and logical components of the argument.

Because of that approach, I encountered some technical issues with the interactive prover functionality. The interactive theorem proving in Lean works similarly to compiling a program in any other programming language. The programmer is writing the code in the editor, while simultaneously the compiler is trying to compile the code which effectively checks the validity of the proof (and incomplete or incorrect proofs result in a compilation error). Consequently, if the proof that we write is correct, then Lean should compile it without producing any errors.

After trying to formalise the proof in one monolithic block of code, I eventually managed to complete it and successfully compile. The main issue I faced was that because of its lack of modularity, the proof was very long and the compilation times took about 5-6 seconds on average. At some point it started slowing down my progress towards the goal, as I couldn't quickly make adjustments to the proof and see the instant response from the compiler.

In order to solve this issue, I needed to introduce multiple short lemmas (e.g. the one below), which aimed at proving facts which are true in a general setting and were used within my main argument. I also isolated the special case of the theorem which was then used to prove the main statement in full generality.

```
/- This lemma asserts that any subset of a closed interval [a,b] is bounded above,
   it is used later to simplify some of the arguments in the special case of the proof. -/
lemma subset_of_Icc_bdd_above {a b : ℝ} {S : set ℝ}
  (h0 : S ⊂ Icc a b) : bdd_above S :=
begin
 use b,
 intros y hy,
 specialize h0 hy,
 exact h0.right,
end
```

An important difference between using Lean to prove a theorem and following a written argument is that certain claims which seem trivial to us might not be so obvious to a computer program. A good illustration of this is the claim below:

$$x \text{ is an upper bound for } S \wedge \forall y \in (m, x] \;\; y \notin S \implies m \text{ is an upper bound for} S.$$

This logical transition was used here in the second part of the proof. In order to be able to use it within my proof, I needed to introduce a new lemma which can be seen in the following code snippet:

```
1  lemma no_elems_lt_upper_bound_imp_better_bound {k m : ℝ} {S : set ℝ}
2    (h0 : m ∈ upper_bounds S) (h1 : ∀ x ∈ Ioc k m, x ∉ S) : k ∈ upper_bounds S :=
3  begin
4    intros a ha,                -- Let a ∈ S be arbitrary.
5    specialize h1 a,            -- Deduce a ∈ (k,m] -> a ∉ S from h1.
6    by_contra,                  -- Argue by contradiction.
7    rw not_le at h,             -- Simplify
8    have hlub : a ∈ Ioc k m,    -- Show a ∈ (k, m] from definition and previous
9    { rw <- Ioc_def,            -- hypotheses.
10     exact ⟨h, h0 ha⟩, },
11   apply h1,                   -- Apply a ∈ (k,m] -> (a ∈ S -> false) to the goal
12   exacts [hlub, ha],          -- Conclude the proof using hypotheses.
13 end
```

The lemma above illustrates how simple facts which seem trivial to human beings require a certain degree of care when proving them formally. Let us try to translage the formal Lean proof into an informal argument to get an understanding of the lemma. In the code snippet above on line 2 we can see the hypotheses that we are given to show the goal: "$k \in$ `upper_bounds` $S$". The first one of them (`h0`) assumes that $m$ is an upper bound for $S$, whereas the second one (`h1`) states that for all elements in the half open interval $(k, m]$, none of them are in $S$. After that, first (on line 4) we unwrap the definition of the set `upper_bounds S` .

By definition $k$ is in that set if for any $a \in S$, we have that $a \leq k$. This is precisely what happens on line 4, we let $a$ be a real number and assume that it belongs to $S$ (that introduces the variable a and the hypothesis `ha:`$a \in S$). After that, we apply the hypothesis `h1` to the assumption $a \in S$ (line 5). This action gives us a hypothesis

$$\texttt{h1} : a \in (k, m] \implies a \notin S.$$

Since previously we have assumed that $a \in S$, it makes sense to try and finish the proof by contradiction (line 6). Next, we perform arithmetic simplifications and then show that given our assumptions we can deduce that $a \in (k, m]$ (lines 7-8). Having shown that, on line 12 we use backwards reasoning. Effectively what happens is that, at that point in the proof, the hypothesis `h1` is internally represented as:

$$a \in (k, m] \implies (a \in S \implies \texttt{false}).$$

Hence, intuitively, given that our aim is to show `false`, if we use `h1` it suffices to show that both $a \in (k, m]$ and $a \in S$ are true and the goal will follow. This is precisely what happens on line 11, our previous goal (`false`) gets replaced by the two separate goals $a \in (k, m]$ and $a \in S$, but those follow directly from our hypotheses, so we conclude the proof (line 12).

Now that we've seen the correspondence between an informal argument and the Lean code, let me go over the structure of the main proof and my approach for formalising it.

In order to prove the IVT first I needed to introduce the definition of continuity as it is the main tool that was used in the informal proof above. I've decided to use a simple $\epsilon - \delta$ definition rather than relying on the mathlib library. That was because, the definition of continuity in mathlib involves topological spaces and it would be a bit difficult to apply it to the simple setting of first year university mathematics. Therefore, I decided to define the notion of continuity as follows:

```
def continuous (f : ℝ → ℝ) : Prop :=
∀ (x : ℝ), ∀ ϵ > 0, ∃ δ > 0, ∀ (y : ℝ), |x- y| < δ → |f x - f y| < ϵ
```

Having introduced this definition I first formulated the special case of the IVT and proved it in a single argument. The way I stated it was the following:

```
theorem intermediate_value_theorem_special {a b : ℝ} {f : ℝ  → ℝ} (h0 : a < b)
(hfcont : continuous f) :
∀ (c : ℝ), c ∈ Ioo (f a) (f b) -> ∃ (x : ℝ), (x ∈ Icc a b) ∧ (f x = c) :=
begin rintro c ⟨hfac, hcfb⟩, ... end
```

One of the most important steps that I needed to justify in the proof of the above theorem, was that a bounded and nonempty set of real numbers has a supremum. At the time of writing the first version of the proof, I didn't have as much experience with searching the mathlib library, and so I couldn't find that particular theorem in there. I tried proving it on my own but that wasn't successful. Here I wanted to acknowledge the members of the Xena Project discord server who helped me find the required statement in the library, the usage of it can be seen below:

```
-- Now we argue that S has a supremum as it is nonempty and bounded.
obtain ⟨x, hxlub⟩ : ∃ (x : ℝ), is_lub S x,
{ apply real.exists_is_lub S hSnoe hba, },
```

An important observation that I made when proving the theorem was that surprisingly the parts of the proof that were the most fiddly and required writing multiply nested arguments were the ones that in a regular proof can be trivially shown using simple arithmetic operations. For instance, the snippet below aims to show that if we have $\delta > 0$ and $x < b$ and if we let $y := x + \frac{1}{2}\min(\delta, b - x)$, then $y < b$. This seems relatively simple in a written argument, however translating that to Lean was more difficult than expected. I needed to explicitly show that $\min \delta(b - x) \leq (b - x)$ which required using the library.

```
change x + 1 / 2 * min δ (b - x) ≤ b,
have hmin: (min δ (b - x)) ≤ (b - x),
{ rw min_le_iff, simp, },
linarith, }, },
```

After proving the special case of IVT, I was expecting the case analysis in the proof of the IVT in full generality to be relatively straightforward. However, it turned out that there was a number of issues that I needed to think about before being able to fully generalise my argument. Firstly, if we look at the statement of the IVT, we can see the hypothesis: "for all $c$ **between** $f(a)$ and $f(b)$". That notion of being "between" turned out to be more complicated than anticipated. I tried searching through the library to see if there is a proposition which conveniently takes three numbers and states that the first one is between the two others. However, after not being able to find anything applicable, I decided to define it myself. The obvious way of stating that is as follows:

$$a \text{ is between } b \text{ and } c \iff a \in [\min(b, c), \max(b, c)].$$

I started by formulating the statement of the theorem using this definition, by replacing the "is between" with the $c \in [\min(f(a), f(b)), \max(f(a), f(b)]$. However, that definition made it less readable as it wasn't immediately obvious what it represented. I thought it would be very convenient to be able to write

`c between f a and f b.`

directly in my code. In order to achieve this, I added the following two definitions:

```
def is_between (a b c : ℝ) : Prop :=
a ∈ Icc (min b c) (max b c)
def is_strictly_between (a b c : ℝ) : Prop :=
a ∈ Ioo (min b c) (max b c)
```

However that wasn't a perfect solution either as using the definitions above would require typing `is_between a b c` and one has to remember which argument is the one that lies between. I wanted to be able to represent it with an infix notation, therefore I introduced the following custom definition:

```
reserve infix ` between `:65
reserve infix ` strictly_between `:65
reserve infix ` and `:66
notation a ` between ` b ` and ` c := is_between a b c
notation a ` strictly_between ` b ` and ` c := is_strictly_between a b c
```

With that notation in place, I formulated the statement of the theorem without losing any readability:

```
theorem intermediate_value_theorem (a b : ℝ) (f : ℝ  → ℝ)
  (h0 : a < b) (hfcont : continuous f) :
  ∀ (c : ℝ), (c between (f a) and (f b)) -> ∃ (x : ℝ), (x ∈ Icc a b) ∧ (f x = c) :=
begin ... end
```

The new notation was also very useful within the proof itself, as it allowed me to keep the code concise and maximise readability. An example of this can be seen below:

```
    intro hIcc,
    have hIoo : c strictly_between (f a) and (f b),
    { let minimum := (min (f a) (f b)),
      let maximum := (max (f a) (f b)), ...},
```

An interesting observation that I made while formalising the case analysis in the body of the main proof was that it seemed fairly straightforward in the informal argument, whereas it required a certain degree of caution in the formal proof in Lean. I needed to first consider the case when either $f(a) = c$ or $f(b) = c$, that was almost a direct translation from the informal argument.

```
  -- First we need consider the case when c is equal to value at either of the endpoints.
  by_cases (c = f a ∨ c = f b),
  { -- Here we use x := a and x := b respectively and show that the required f(x) = c. }
```

However, after that, the other case was generated by Lean and was the exact negation of the first one, so I had the hypothesis $\neg(f(a) = c \text{ or } f(b) = c)$ and needed to somehow transform it into either $f(a) < c < f(b)$ or $f(b) < c < f(a)$. That simple transition required another search through the mathlib library. In the end I used the trichotomy of "<" on $f(a) < f(b)$ and considered three resulting cases separately:

```
    -- Here we consider the trichotomy : (f(a) < f(b)) ∨ (f(a) = f(b)) ∨ (f(b) < f(a)).
    have htri := lt_trichotomy (f a) (f b),
    -- The rest of the proof is split into 3 cases.
    rcases htri with hlt | heq | hgt,
```

That resulted in a nested structure of the case analysis. One thing that I could possibly work on in the future would be simplifying that argument so that it more closely resembles the informal case analysis.

## Conclusions

Summarising the observations listed above, the proof that I formalised in the first part of my coursework allowed me to gain experience proving statements involving real analysis. I have learned not to judge the difficulty of formalising a statement by how simple it is to prove it informally. The project gave me an opportunity to appreciate the importance of breaking the proof down into separate lemmas in order to simplify the argument as well as improve the performance of the compiler while checking the proof. I also had an opportunity to explore more advanced aspects of programming in Lean such as introducing the new infix notation to encapsulate the notion of "being between" two numbers.