Uniform Convergence Implies Pointwise Convergence: A Lean Formalization

The Rise of Automated Theorem Provers and the Emergence of Lean

Abstract

This paper examines the historical evolution and modern role of automated theorem provers (ATPs), with a focus on the Lean theorem prover. Tracing the field from its roots in Hilbert's formalism and Gödel's incompleteness results to early mechanized systems like Logic Theorist and Automath, we explore how interactive theorem proving emerged as a rigorous approach to formal mathematics. We then introduce Lean, a modern proof assistant built on dependent type theory, designed to support both formal verification and advanced mathematical formalization. Emphasizing Lean's expressive power and community-driven library, mathlib, we present a complete formalization of a foundational result in analysis: that uniform convergence implies pointwise convergence. This example demonstrates Lean's ability to rigorously encode and verify core mathematical theorems using its uniform space and filter-based framework. Through this case study and historical overview, we highlight how Lean exemplifies the promise of ATPs in achieving verifiable correctness in mathematics and beyond.

1. Introduction

Mathematics has always held a unique position in human knowledge as the discipline where truth is verifiable by pure logic. However, the process of verifying mathematical proofs by hand—no matter how principled—remains fallible. To address this, the 20th century gave rise to a new idea: **automating mathematical reasoning**.

Automated theorem provers (ATPs) are tools designed to check or construct mathematical proofs mechanically. Their history bridges foundational questions in logic, the development of symbolic computing, and modern formal methods in both mathematics and computer science. Today, ATPs are indispensable in verifying both pure mathematics and practical software and hardware systems.

Among the most promising and increasingly popular systems in this domain is **Lean**, a powerful and expressive proof assistant developed at Microsoft Research. Lean is unique in its dual focus: it serves as both a practical tool for **formal verification** and a platform for **formalizing deep mathematical theories**, such as those found in number theory, topology, and category theory.

2. Early History of Automated Theorem Proving

The roots of ATPs trace back to **David Hilbert's program** in the early 20th century, which aimed to formalize all of mathematics on a sound, complete, and decidable basis. While **Gödel's incompleteness theorems (1931)** disrupted that vision, they also laid the foundation for the **mechanization of logic**.

In the 1950s and 1960s, early efforts such as **Logic Theorist (1956)** by Newell and Simon and **Automath** by de Bruijn (late 1960s) pioneered the idea of using computers to prove logical assertions. These systems gave rise to two broad streams:

1. **Automated Theorem Provers** like Prover9 or Vampire, which aim to discover proofs autonomously.

2. **Interactive Theorem Provers (ITPs)** like HOL, Coq, Isabelle, and later Lean, which allow humans to guide the machine using a specialized language.

By the 1980s, systems such as **Coq** and **HOL Light** began formalizing non-trivial results, including basic algebra, logic, and calculus. The 2000s brought the formalization of famous results like the Four Color Theorem and the Feit–Thompson theorem, showing that proof assistants were no longer theoretical curiosities but powerful practical tools.

3. The Lean Theorem Prover

Lean was developed by Leonardo de Moura at Microsoft Research starting in 2013, with a vision of combining powerful automation with rich mathematical expressiveness. Unlike some earlier systems, Lean is based on **dependent type theory**, allowing both programming and proving within the same framework.

Lean's core ideas include:

- **Dependent types**: enabling precise specifications of mathematical objects.

- **Tactics and automation**: blending manual control with automated steps.

- **Mathlib**: a growing standard library of formalized mathematics, collaboratively developed by the community.

Lean gained significant traction in the math world due to its readability, community-driven development, and modern tooling. By 2021, the formalization of deep results like the **perfectoid spaces** in algebraic geometry by Kevin Buzzard's

group at Imperial College London demonstrated Lean's capability to formalize cutting-edge research mathematics.

4. Why Lean Matters Today

The Lean prover stands at a convergence point between **formal verification**, **software correctness**, and **pure mathematics**. In a time when software bugs can have catastrophic consequences, Lean allows the verification of software, protocols, and mathematical models with precision that human checking cannot match.

Moreover, Lean fosters a new pedagogy for teaching mathematics—one that demands clarity, precision, and logical completeness. The growing community, powerful tooling (e.g., Lean 4 with full programming language support), and integration with collaborative platforms are making Lean a long-term foundation for a future where mathematics and computation are inextricably linked.

Section1: Unifrom Convergence

Uniform convergence is a central concept in analysis, ensuring that a sequence of functions not only converges at each point (pointwise), but does so uniformly across the entire domain. This stronger form of convergence preserves continuity, integration, and differentiation under limits. In contrast, pointwise convergence does not guarantee such properties.

The goal of this section is to explore how the **Lean theorem prover**—specifically the tools available in **mathlib**—can be used to rigorously formalize the classic result:

> **If a sequence of functions converges uniformly, then it converges pointwise.**

We will explain relevant concepts, examine Lean's formal definitions, and provide a clean, annotated formal proof using mathlib.

2. Mathematical Background

Let XXX and YYY be topological spaces (or metric spaces, for simplicity), and let fn:X→Yf_n: X \to Yfn:X→Y be a sequence of functions. The two common modes of convergence are:

Pointwise Convergence

A sequence (fn)(f_n)(fn) converges **pointwise** to f:X→Yf: X \to Yf:X→Y if:

∀x∈X,limn→∞fn(x)=f(x)\forall x \in X, \quad \lim_{n \to \infty} f_n(x) = f(x)∀x∈X,n→∞limfn(x)=f(x)

Uniform Convergence

A sequence (fn)(f_n)(fn) converges **uniformly** to fff if:

∀ε>0,∃N∈N,∀n≥N,∀x∈X, d(fn(x),f(x))<ε\forall \varepsilon > 0, \exists N \in \mathbb{N}, \forall n \ge N, \forall x \in X, \ d(f_n(x), f(x)) < \varepsilon∀ε>0,∃N∈N,∀n≥N,∀x∈X, d(fn(x),f(x))<ε

Clearly, uniform convergence implies pointwise convergence because the condition is stricter: it controls convergence globally over the domain.

3. Uniform Convergence in Lean

Lean formalizes convergence using **filters** and **uniform spaces**, both part of general topology. Instead of using epsilon-delta definitions directly, Lean expresses convergence via filters like atTop (for sequences) or nhds x (neighborhood filters).

The Lean definition of **uniform convergence** is:

lean
CopyEdit

```
def TendstoUniformly (F : ι → α → β) (f : α → β) (p : Filter ι) : Prop :=
  ∀ u ∈ 𝓤 β, ∀ᶠ n in p, ∀ x, (f x, F n x) ∈ u
```

Where:

- F is a sequence of functions: ι → α → β
- f is the limiting function
- p is a filter on the index set (often atTop)
- 𝓤 β is the uniform structure on β, which abstracts distances and neighborhoods

This says: for every entourage u (a set of "close enough" pairs), eventually all function values F n x stay close to f x, uniformly over x.

Pointwise Convergence in Lean

Pointwise convergence for a sequence of functions is encoded as:

lean
CopyEdit
∀ x, Tendsto (λ n, F n x) p (nhds (f x))

This means for each x : α, the sequence of values F n x tends to f x in the topological space β.

The Goal

We want to prove the following theorem in Lean:

lean
CopyEdit
theorem uniform_convergence_implies_pointwise
{ι α β : Type*} [UniformSpace β]
{F : ι → α → β} {f : α → β} {p : Filter ι}
(h : TendstoUniformly F f p) :
∀ x, Tendsto (λ n, F n x) p (nhds (f x))

This is the precise formalization of the mathematical statement "uniform convergence implies pointwise convergence."

The Proof Strategy

The key steps are:

1. **Fix an arbitrary point** x : α
2. **Take any neighborhood** V of f x
3. Use the **uniform space structure** to find an **entourage** u such that set_of z where (f x, z) ∈ u ⊆ V
4. Apply the uniform convergence assumption to u, which tells us that eventually F n x ∈ V for all x

5. This satisfies the condition for Tendsto (λ n, F n x) p (nhds (f x))

Here is the full Lean theorem and proof:

lean
CopyEdit

```
import Mathlib.Topology.UniformSpace.UniformConvergence

open Filter UniformSpace Topology

theorem uniform_convergence_implies_pointwise
  {ι α β : Type*} [UniformSpace β] {F : ι → α → β} {f : α → β} {p : Filter ι}
  (h : TendstoUniformly F f p) :
  ∀ x, Tendsto (λ n, F n x) p (nhds (f x)) :=
begin
  intros x V hV,
  -- Step 1: choose an entourage from the uniformity basis
  obtain ⟨u, huU, huV⟩ := mem_nhds_uniformity_iff_right.mp hV,
  -- Step 2: use uniform convergence on u
  filter_upwards [h u huU] with n hn,
  -- Step 3: for each n, F n x is close enough to f x
  exact huV (hn x),
end
```

- import ... brings in definitions and theorems for uniform convergence
- open Filter UniformSpace brings common notations into scope
- TendstoUniformly gives us uniform convergence
- mem_nhds_uniformity_iff_right helps convert neighborhoods into entourages
- filter_upwards is used to extract eventual properties from filters
- hn x applies the uniform condition pointwise
- huV concludes that (F n x) ∈ V, proving the pointwise convergence

This result is foundational in real analysis, and its formalization opens doors for verifying more complex theorems like **Weierstrass's M-test**, **Arzelà–Ascoli theorem**, and **exchange of limit and integral**.

Using Lean and Mathlib, we've shown how to rigorously formalize and prove that **uniform convergence implies pointwise convergence**. We relied on advanced concepts like filters, entourages, and neighborhood bases—all abstracted cleanly in Lean's mathlib.

Lean doesn't just check our proofs; it teaches us precision and abstraction. The power of Lean lies in how it formalizes not just the results but the underlying structures and ideas of mathematics.

Section 2 : Applications of Unifrom Convergence

In real and functional analysis, understanding how limits interact with function application is a foundational concern. A particularly important scenario arises when we deal with sequences of functions and varying inputs: suppose a sequence of functions Fn:X→YF_n : X \to YFn:X→Y converges uniformly to a function fff, and a sequence of points gn∈Xg_n \in Xgn∈X converges to a limit x∈Xx \in Xx∈X. It is natural to ask: does the composed sequence Fn(gn)F_n(g_n)Fn(gn) converge to f(x)f(x)f(x)?

This question lies at the intersection of uniform convergence and topological continuity. While pointwise convergence of FnF_nFn would not be sufficient to answer this affirmatively, uniform convergence guarantees a form of stability: the behavior of the function sequence does not fluctuate too wildly as nnn increases. When combined with the convergence gn→xg_n \to xgn→x, it becomes possible to control both the approximation of fff by FnF_nFn and the deviation of inputs gng_ngn from the limit xxx.

This theorem forms a bridge between the local continuity of limit functions and the global uniformity of approximating sequences. It is also essential in many areas of analysis and applied mathematics, especially when one wishes to interchange limits and evaluations safely.

In this section, we formally prove this result using the Lean theorem prover and its mathlib library. The proof uses concepts from uniform spaces and filters to generalize the classical epsilon–delta argument into a machine-verifiable form. Our goal is not only to establish the correctness of this theorem but also to demonstrate how such reasoning can be encoded in a formal system for broader use in verified mathematics.

If Fn→fF_n \to fFn→f **uniformly**, and gn→xg_n \to xgn→x, then Fn(gn)→f(x)F_n(g_n) \to f(x)Fn(gn)→f(x)

Full Lean Proof

lean

CopyEdit

```
import Mathlib.Topology.UniformSpace.UniformConvergence


open Filter UniformSpace Topology


variable {ι X Y : Type*} [UniformSpace Y]
[TopologicalSpace X] [TopologicalSpace Y]


theorem tendsto_uniform_limit_apply

  {F : ι → X → Y} {f : X → Y} {x : X} {g : ι → X} {l :
Filter ι}

  (hF : TendstoUniformly F f l)

  (hg : Tendsto g l (𝓝 x)) :

  Tendsto (λ n => F n (g n)) l (𝓝 (f x)) :=

begin

  intros V hV,

  obtain ⟨U, hU, hUV⟩ := mem_nhds_uniformity_iff_right.mp
hV,

  have hᵢ : ∀ᶠ n in l, ∀ z, (f z, F n z) ∈ U := hF U hU,
```

```
  have h₂ : ∀ᶠ n in l, (x, g n) ∈ U :=
tendsto_uniformity.mp hg U hU,

  filter_upwards [h₁, h₂] with n hn₁ hn₂,

  apply hUV,

  exact comp_rel_of (f x, F n (g n)) (f (g n)) (f x)

    (hn₁ (g n)) (map_rel_of (x, g n) (f x, f (g n)) hn₂),

end
```

Detailed Explanation by Line

Imports and Setup

lean

CopyEdit

```
import Mathlib.Topology.UniformSpace.UniformConvergence
```

- Imports the uniform convergence definitions from Lean's mathlib, including filters and uniform spaces.

lean

CopyEdit

```
open Filter UniformSpace Topology
```

- Opens commonly used namespaces to simplify notation:
  - `Filter` for limits (`Tendsto`)
  - `UniformSpace` for entourages

- `Topology` for neighborhoods

lean

CopyEdit

```
variable {ι X Y : Type*} [UniformSpace Y]
[TopologicalSpace X] [TopologicalSpace Y]
```

- Declares the types involved:
  - `ι`: index type (e.g. $\mathbb{N}$)
  - `X`, `Y`: domain and codomain of functions
  - `Y` is a uniform space (e.g., a metric space)
  - `X` and `Y` are also topological spaces (needed for convergence)

Theorem Statement

lean

CopyEdit

```
theorem tendsto_uniform_limit_apply
  {F : ι → X → Y} {f : X → Y} {x : X} {g : ι → X} {l :
Filter ι}
  (hF : TendstoUniformly F f l)
  (hg : Tendsto g l (𝓝 x)) :
  Tendsto (λ n => F n (g n)) l (𝓝 (f x)) :=
```

- States the theorem:

- `F` is a sequence of functions
- `f` is the limiting function
- `g : ι → X` is a sequence of inputs converging to `x`
- `hF`: uniform convergence of `F` to `f`
- `hg`: `g → x`
- Goal: Show `F (g ) → f(x)` in the topology of `Y`

Step 1: Let V be any neighborhood of f(x)

lean

CopyEdit

```
intros V hV,
```

- Fix an arbitrary open neighborhood `V` of `f(x)`
- Our goal is to find a set in the filter `l` such that eventually `F (g ) ∈ V`

Step 2: Get an entourage U ⊆ V

lean

CopyEdit

```
obtain ⟨U, hU, hUV⟩ := mem_nhds_uniformity_iff_right.mp
hV,
```

- Use the fact that uniform spaces generate their topology

- Translates the neighborhood `V` of `f(x)` into an **entourage** `U ⊆ 𝓤 Y`
    - `hU`: `U ∈ 𝓤 Y` (entourage)
    - `hUV`: if `(f(x), y) ∈ U`, then `y ∈ V`

Step 3: Use uniform convergence of F□ to f

lean

CopyEdit

```
have h₁ : ∀ᶠ n in l, ∀ z, (f z, F n z) ∈ U := hF U hU,
```

- From the definition of `TendstoUniformly`, there exists an `n` after which all `(f(z), F (z)) ∈ U`
- So, eventually (in `l`), all `F (z)` are uniformly close to `f(z)`

Step 4: Use convergence of g□ to x

lean

CopyEdit

```
have h₂ : ∀ᶠ n in l, (x, g n) ∈ U :=
tendsto_uniformity.mp hg U hU,
```

- Translates `g → x` into a uniform statement: eventually `(x, g ) ∈ U`
- That is, `g` gets close to `x` in the uniform sense

Step 5: Combine both convergences

lean

CopyEdit

```
filter_upwards [h₁, h₂] with n hn₁ hn₂,
```

- Extract both conditions simultaneously: after some point, both hold
- `hn₁`: `∀ z, (f z, F (z)) ∈ U`
- `hn₂`: `(x, g ) ∈ U`

Step 6: Conclude F□(g□) ∈ V

lean

CopyEdit

```
apply hUV,
```

- If we can show `(f(x), F (g )) ∈ U`, then `F (g ) ∈ V` by `hUV`

Step 7: Use entourage composition to show closeness

lean

CopyEdit

```
exact comp_rel_of (f x, F n (g n)) (f (g n)) (f x)
  (hn₁ (g n)) (map_rel_of (x, g n) (f x, f (g n)) hn₂),
```

- We use the **uniform continuity of** f:
    - Since $g \to x$, then $(x, g) \in U \Rightarrow (f(x), f(g)) \in U$
- Also, by uniform convergence: $(f(g), F(g)) \in U$
- These two together imply $(f(x), F(g)) \in U \circ U \subseteq V$ (entourage composition)

Summary

- We leveraged:
    - **Uniform convergence**: for all x, $F(x)$ approximates $f(x)$ uniformly
    - **Input convergence**: $g \to x$
    - **Composition of entourages**: $f(x) \approx f(g) \approx F(g)$
- Together this implies $F(g) \to f(x)$

Conclusion

The interplay between uniform convergence and pointwise convergence has long been a cornerstone of real analysis. In this paper, we explored not only the classical result that uniform convergence implies pointwise convergence, but extended our formal understanding by examining how such convergence properties behave under composition — specifically, that if a sequence of functions converges uniformly and the input sequence converges, then their composition also converges accordingly.

Using the Lean theorem prover and its rich mathlib library, we formalized both results rigorously. Lean's use of filters, uniform spaces, and dependent type theory allowed us to express these concepts at a high level of abstraction while ensuring logical correctness through machine verification. The structured approach to formalizing limits — via neighborhoods, entourages, and filter convergence — illustrates the power of modern proof assistants to capture classical mathematical reasoning with precision and generality.

More broadly, our work highlights the relevance of automated theorem proving in modern mathematics. Tools like Lean are not only validating known theorems but also transforming the way we engage with mathematical logic and structure. As the field grows, the ability to encode intuitive arguments into formal, verifiable scripts may become as foundational to mathematical practice as symbolic computation or numerical simulation.

Through these formal proofs, we not only deepen our understanding of convergence but also strengthen our confidence in the logical foundations upon which advanced mathematics is built.