Theorems on Hausdorff Spaces

Ryland Gross

June 12, 2026

Abstract

In this final project for Math 161, I present a variety of theorems on Hausdorff spaces, three of which are exercises from Munkres and the remainer theorems proven in his textbook. The goal of this project is not merely to formalize something I already know but to present it in a fashion suitable to students. This project exposes the logic in what are often very wordy proofs, including certain steps the reference textbook skips over or treats as trivialities. We demonstrate arguments such as: separating points by open neighborhoods, proving singletons and finite sets are closed, passing the Hausdorff property to subspaces and products, and relating the Hausdorff condition to closedness of the diagonal.

Contents

1 Introduction

Guiding idea. I often see jumps in mathematical reasoning, even in "rigorous" proofs. This is especially prevalent in Topology, where proofs can be wordy. Here we use Lean4 to make sure all careful arguments about complements, closures, subspaces, and finitude become explicit.

While working through the project I picked out five results carefully (and did one twice) to emphasize the different thinking you must use to formalize results in Lean. First, we show that singletons are closed in a Hausdorff space. We do this twice to show how formalization can lead to a more grindy result if you fail to find relevant theorems. Next, we show any finite set must be closed as a corollary. Afterwards, we examine subspace and product topology to demonstrate how to deal with type management in Lean, and finally, we show a classic result on the diagonal using our earlier work.

2 Mathematical Background

2.1 Hausdorff Spaces

A Hausdorff space is a topological space X such that for every $x, y \in X$ such that $x \neq y$ we have open sets U and V where $x \in U$, $y \in V$ and $U \cap V = \emptyset$. Try comparing this natural language definition to the code below.

2.2 Formal Definition in Lean

hausdorff-01

```
1 class TopologyIsHausdorff (X : Type u) [TopologicalSpace X] : Prop where
2   /-- Every two points in a Hausdorff space admit disjoint open neighbourhoods. -/
3   t2 : Pairwise fun x y =>
4     ∃ U V : Set X, IsOpen U ∧ IsOpen V ∧ x ∈ U ∧ y ∈ V ∧ Disjoint U V
```

```
=== LeanTeX Status ===
[info] 1:1: Lean accepted this snippet.
=== Infoview Messages ===
[info] 1:1: No diagnostics or tactic state were produced.
```

This definition is directly pulled from Mathlib, and maintains our earlier meaning. Throughout the paper you will see the last condition as *Disjoint*. In Lean this definition is written as T2Space.

3 Core Formalized Results

3.1 Singletons are Closed

Mathematical statement For a Hausdorff space X with $x \in X$ we know that the singleton set $\{x\}$ is closed.

Proof: As a sketch, to start we will just prove that the complement is open. We do this by showing that for every $a \in \{x\}^c$ there is an open set not intersecting $\{x\}$. First, assume that there is some $a \notin \{x\}$. (We must prove the case that there is only $\{x\} = X$ also) Now, apply Hausdorff to a and x. This means we have $a \in U$, $x \in V$ and $U \cap V = \emptyset$. and since $a \in U$ does not intersect $\{x\}$ since this is a subset of V it is disjoint from U. Thus we have satisfied the first condition we had.

hausdorff-02

```
theorem singletonisclosed2 (X : Type u) [TopologicalSpace X] (x : X) [h : TopologyIsHausdorff X]
↪  : IsClosed ({x} : Set X) := by
  rcases h with ⟨h⟩
  refine ⟨?_⟩
  rw [isOpen_iff_forall_mem_open]
  intro a ha
  simp only [Set.mem_compl_iff] at ha
  have : x ≠ a := by
    exact Ne.intro fun a_1 => ha (id (Eq.symm a_1))
  have := h this
  simp at this
  rcases this with ⟨u, hu, v, hv, xinu, ainv, disj⟩
  use v
  have : x ∉ v := by
    exact Disjoint.notMem_of_mem_left disj xinu
  constructor
  · exact Set.subset_compl_singleton_iff.mpr this
  constructor
  · exact hv
  apply ainv
```

```
=== Infoview State (first 5 line(s)) ===
[info] 20:1: [infoview] tactic state:
no goals
=== Infoview Messages ===
[info] 1:1: [infoview message] Goals accomplished!
```

Some Lean Basics in this proof: In the above proof `rcases h` merely allows us to extract what being hausdorff means. `refine` moves the proof's goal to proving that the complement is open the following rewrite `rw` statement is there to move the goal to showing that a set is open if for a general member of the complement there is some open neighborhood around

it not intersecting the original set. Our simp only functions very similarly to rewrite here explaining what the complement is. I highly recommend playing with these in the lean theorem prover to get more exposed to the language and see the proof better.

3.2 Finite Sets are Closed

Mathematical statement Every finite subset of a Hausdorff space is closed.

Proof: For some finite set, we know that it can be broken down into the union of singleton sets. Singleton sets are closed. We know finite unions of closed sets are closed. Thus a finite union of singleton sets is closed. Hence, finite sets are closed.

hausdorff-03

```
theorem finitesetisclosed (X : Type u) [h1 : TopologicalSpace X] [h: TopologyIsHausdorff X] (s :
↪  Set X) (hs : s.Finite) : IsClosed s := by
    have hrewrite : s = ⋃  x ∈ hs.toFinset, ({x} : Set X) := by
      ext y
      simp
    rw [hrewrite]
    refine isClosed_biUnion_finset ?_
    intro i hi
    apply singletonisclosed
```

```
=== Messages ===
[error] 8:10: Unknown identifier `singletonisclosed`
=== Infoview State (first 5 line(s)) ===
[info] 9:1: [infoview] tactic state:
no goals
```

Why is this proof so short?: It isn't! We just did most of the heavy lifting before by proving that singletons are closed. Thus, when we arrived at the end proof state in this theorem applying that theorem `apply singletonisclosed` closed out proof state and we typechecked the proof. It is remarkable then that we do have some work to do first since in Munkres he writes that it suffices to show that singletons are closed. But here we first must engage with finsets and fix our union argument before we enter the proof he is talking about.

3.3 Subspaces of Hausdorff Spaces are Hausdorff

Mathematical statement Have Hausdorff X and $s \subseteq X$, then the subspace s is Hausdorff.

Proof: Since s is a subset of X take some points $x, y \in s \subset X$ and assume that they are not equal. Now my the hausdorff condition on X you know that there exist open sets U and V

disjoint such that $x \in U$ and $y \in V$ now intersect these with s and in the subspace topology they are still open and the rest of the argument follows.

hausdorff-04

```
theorem subspacehdishd (X : Type u) [h: TopologicalSpace X] [h1: TopologyIsHausdorff X] (s : Set
↪  X) : TopologyIsHausdorff s := by
  refine ⟨?_⟩
  intro x y hxy
  simp
  rcases h1 with ⟨h1⟩
  simp at h1
  have xneyinx: (x: X) ≠ (y : X) := by
    grind
  have := h1 xneyinx
  simp at this
  rcases this with ⟨u, hu, v, hv, xinu, yinv, disj⟩
  -- Now we want to take intersections with s such that these inherit the subspace topology
  use Subtype.val ⁻¹' u -- (this is equivalent to the intersection with s)
  constructor
  · exact hu.preimage continuous_subtype_val -- so instead of using subspace topology we know
  ↪  that this preimage is a cont map
  use Subtype.val ⁻¹' v
  constructor
  · exact hv.preimage continuous_subtype_val
  constructor
  · trivial
  constructor
  · trivial
  exact Disjoint.preimage Subtype.val disj
```

```
=== Infoview State (first 5 line(s)) ===
[info] 24:1: [infoview] tactic state:
no goals
=== Infoview Messages ===
[info] 1:1: [infoview message] Goals accomplished!
```

Some Lean Topology: Here we see some weird argument using maps. To make this a little more intuitive we need to think about the subspace topology with some lean intuition. In Lean, you are dealing with two types `x : Set S` and `x: Set X` in this proof. Here I have tried to make that go away as quickly as possible by using `grind` in line 8. Essentially we need to first consider a map that continuously maps between S and X which in regular topology is obvious for the subspace topology, we just know $S \cap X$ will have open sets $U \cap X$ and $V \cap X$ but we actually need this map in lean `subtype_val` to do the heavy lifting for us.

3.4 Products of Hausdorff Spaces are Hausdorff

Mathematical statement If X and Y are Hausdorff spaces, then $X \times Y$ is Hausdorff.

Proof idea First take $x = (x_1, x_2)$ and $y = (y_1, y_2)$ where $x \neq y$ in $X \times Y$. Then either $x_1 \neq y_1$ or $x_2 \neq y_2$. We break into cases. If $x_1 \neq y_1$, we apply that X is Hausdorff and get that there exist disjoint open sets $U, V \subseteq X$ with $x_1 \in U$ and $y_1 \in V$. Then $U \times Y$ and $V \times Y$ are open in $X \times Y$, contain x and y respectively, and remain disjoint because $(U \cap V) \times Y = \emptyset$.If $x_2 \neq y_2$, the same argument as before but in Y applies. In both cases, there are disjoint open neighborhoods of x and y, so $X \times Y$ is Hausdorff. For full Lean code see appendix.

hausdorff-05

```
1  theorem prodhdishd (X : Type u) (Y : Type v) [TopologicalSpace X] [TopologicalSpace Y] [hhausx
   ↪  : TopologyIsHausdorff X] [hhausy : TopologyIsHausdorff Y] : TopologyIsHausdorff (X × Y)
   ↪  := by
```

```
=== Messages ===
[error] 1:188: unsolved goals
X : Type u
Y : Type v
inst✝¹ : TopologicalSpace X
inst✝ : TopologicalSpace Y
hhausx : TopologyIsHausdorff X
hhausy : TopologyIsHausdorff Y
⊢ TopologyIsHausdorff (X × Y)
[error] 1:190: unexpected token 'theorem'; expected '{' or tactic
=== Infoview State (first 5 line(s)) ===
[info] 2:1: [infoview] tactic state:
X : Type u
Y : Type v
inst✝¹ : TopologicalSpace X
inst✝ : TopologicalSpace Y
... (3 more line(s) hidden)
```

Why is this proof so long in lean?: We need to actually use every case even if it is WLOG the exact same as earlier, which we use in the english proof. This is just as easy as copying and pasting in lean, but it does end up giving us a much longer and more complicated proof.

3.5 The Diagonal Criterion

Mathematical statement If X is Hausdorff then we have that the diagonal $\Delta_X = \{(x, x) : x \in X\} \subseteq X \times X$ is closed.

Proof sketch It suffices to prove that in the complement of the diagonal every point has an open set not intersecting the diagonal. Choose some (a, b) is outside the diagonal, then $a \neq b$. By Hausdorffness, choose disjoint open neighborhoods U of a in X and V of b in X. Then $U \times V$ is an open neighborhood of (a, b) disjoint from the diagonal.

hausdorff-06

```
theorem diagonal_isClosed (X : Type u) [h: TopologicalSpace X] [h2: TopologyIsHausdorff X] :
↪  IsClosed {p : X × X | p.1 = p.2} := by
    refine ⟨?_⟩
    have ishausx := h2
    rcases h2 with ⟨h1⟩
    simp at h1
    have := prodhdishd X X
    rcases this with ⟨h3⟩
    simp at h3
    -- Suffices to show that this works for arbitrary point in the complement
    rw [isOpen_iff_forall_mem_open]
    intro q hq
    -- have complement of {p | p.1 = p.2} is {q | q.1 ≠ q.2}
    simp only [Set.mem_compl_iff, Set.mem_setOf_eq] at hq
    push Not at hq
    -- have in X there are open sets u and v such that q.1 ∈ u q.2 ∈ v and disjoint u v by
    ↪  hausdorffx
    have := h1 hq
    simp at this
    rcases this with ⟨u, hu, v, hv, q1inu, q2inv, disj⟩
    use (u ×ˢ v)
    constructor
    · intro x hx
      rcases hx with ⟨h1, h2⟩
      have : x.1 ≠ x.2 := by
        exact Disjoint.ne_of_mem disj h1 h2
      simp [this]
    constructor
    · exact IsOpen.prod hu hv
    rw [Set.mem_prod]
    constructor
    · exact q1inu
    exact q2inv
```

```
=== Infoview State (first 5 line(s)) ===
[info] 32:1: [infoview] tactic state:
no goals
=== Infoview Messages ===
[info] 1:1: [infoview message] Goals accomplished!
```

4 Formalization Reflection

I started this project because I often try to formalize little things from textbooks in my free time. What was remarkable to me was this theorem from Munkres, and how many steps are skipped over in writing his proofs. Consider the following proof,

Theorem 17.8. Every finite point set in a Hausdorff space X is closed.

Proof. It suffices to show that every one-point set $\{x_0\}$ is closed. If x is a point of X different from x_0, then x and x_0 have disjoint neighborhoods U and V, respectively. Since U does not intersect $\{x_0\}$, the point x cannot belong to the closure of the set $\{x_0\}$. As a result, the closure of the set $\{x_0\}$ is $\{x_0\}$ itself, so it is closed.

There is actually a fair amount of work to be done in this line, *Since U does not intersect $\{x_0\}$, the point x cannot belong to the closure of the set $\{x_0\}$.* In Lean one of the ways that I did this initially was to use monotonicity of closure, which you can see here in our lean proofs. So it was my goal to make more of these steps explicit.

I did however, notice a tendency while working on the project that I would sometimes try to grind a proof without trying simplification steps or `apply?`. For example, I exhibited a proof of `singletonisclosed` by showing the closure of $\{x\}$ is itself and did this using set theory and monotonicity of closure. You can see this proof in the appendix.

But this proof isn't elegant, and while it might show real engagement both with the verifier and the topology it fails to be informational. I think I have a tendency while working with lean not to write my own proofs first, and so I wound up finding some way to fight doing both that and reading more of the documentation of topology on mathlib. I managed to read more of it when I needed to for the remaining proofs which is why I went back and reproved our singleton proof.

Another problem I ran into was my lack of understanding of type. In particular working on different topological spaces and sets its often hard to get through that part, but `grind` managed to. Additionally I noticed that the subspace topology works far differently because it is defined over maps with inverses so that you can recover certain facts. Once I used these from the docuemntation `apply?` managed to work just fine.

5 Some Practice

1. Prove the reverse direction of the diagonal theorem.
2. Prove that compact subsets of Hausdorff spaces are closed.
3. Prove the uniqueness of limits in Hausdorff spaces.

6 Conclusion

Now, what I learned from this is that there is sincere need for introductions to the documentation for any work we might want to formalize. I doubt this would be hard for the mathlib community to make which is why its remarkable to me that it does not already exist.

Additionally, I think as I treat lean as a hobby it must be accompanied by natural proof since it is so easy to just grind through things inefficiently in the IDE on its own. I think as a reflection from the class while this has helped me think of proof in a more structured way

there is some concern in me that it has also made me a bit less creative in my proofs that I wrote for class.

Finally, I am deeply grateful to have learned how to use Lean. My only regret about learning it is that I did not take the time to learn it as soon as I knew about mathematics.

Appendix

Product of Hausdorff Spaces is Hausdorff

hausdorff-07

```
theorem prodhdishd (X : Type u) (Y : Type v) [TopologicalSpace X] [TopologicalSpace Y] [hhausx
↪  : TopologyIsHausdorff X] [hhausy : TopologyIsHausdorff Y] : TopologyIsHausdorff (X × Y)
↪  := by
-- this part is all just getting it into a place where we can work with it
refine ⟨?_⟩
rcases hhausx with ⟨hausdorffx⟩
rcases hhausy with ⟨hausdorffy⟩
simp only [exists_and_left] at hausdorffx
simp only [exists_and_left] at hausdorffy
simp only [exists_and_left]
intro x y xney
-- the next part is a triviality but we use grind to save time finding the proof
have : x.1 ≠ y.1 ∨ x.2 ≠ y.2 := by
  grind
rcases this with h1 | h2
· have := hausdorffx h1
  simp at this
  rcases this with ⟨u, hu, v, hv, x1inu, y1inv, disj⟩
  use u ×ˢ (Set.univ : Set Y)
  constructor
  · apply isOpen_prod_iff'.mpr
    constructor
    · constructor
      · assumption
      exact isOpen_univ
  use v ×ˢ (Set.univ : Set Y)
  constructor
  · apply isOpen_prod_iff'.mpr
    constructor
    · constructor
      · assumption
      exact isOpen_univ
  constructor
  · trivial
  constructor
  · trivial
  exact Set.Disjoint.set_prod_left disj Set.univ Set.univ
have := hausdorffy h2
simp at this
rcases this with ⟨u, hu, v, hv, x1inu, y1inv, disj⟩
use (Set.univ : Set X) ×ˢ u
constructor
· apply isOpen_prod_iff'.mpr
  constructor
  · constructor
    · exact isOpen_univ
    assumption
use (Set.univ : Set X) ×ˢ v
constructor
```

```
  · apply isOpen_prod_iff'.mpr
    constructor
    · constructor
      · exact isOpen_univ
      assumption
  constructor
  · trivial
  constructor
  · trivial
  exact Set.Disjoint.set_prod_right disj Set.univ Set.univ
```

```
=== Messages ===
[error] 1:10: `prodhdishd` has already been declared
```

Here is the singleton by monotonicity of closure.

hausdorff-08

```
theorem singletonisclosed (X : Type u) [TopologicalSpace X] (x : X) [h : TopologyIsHausdorff
↪  X] : IsClosed ({x} : Set X) := by
rcases h with ⟨h⟩
by_cases existsdiffx : ∃ y : X, y ≠ x
· rcases existsdiffx with ⟨y, hy⟩
  have := h hy
  simp only [exists_and_left] at this
  have forallynexynotinclosurex : ∀ y : X, y ≠ x → y ∉ closure ({x} : Set X) := by
    intro y hy
    have := h hy
    simp only [exists_and_left] at this
    rcases this with ⟨u, hu, v, hv, yinu, xinv, disj⟩
    have : Disjoint u ({x} : Set X) := by
      simp only [Set.disjoint_singleton_right]
      exact Disjoint.notMem_of_mem_right disj xinv
    have := Disjoint.closure_right disj hu
    have : ({x} : Set X) ⊆ v := by
      exact Set.singleton_subset_iff.mpr xinv
    have : closure ({x} : Set X) ⊆ closure v := by
      exact closure_mono this
    have : Disjoint (closure {x}) u := by
      (expose_names; exact Disjoint.closure_left (id (Disjoint.symm this_2)) hu)
    exact Disjoint.notMem_of_mem_left (id (Disjoint.symm this)) yinu
  have closureeqself : closure ({x} : Set X) = ({x} : Set X) := by
    by_contra! h
    have notemptyclosure : closure ({x} : Set X) ≠ ∅ := by
      simp
    have := helperforsingleton x (closure ({x} : Set X)) h notemptyclosure
    rcases this with ⟨a, ha1, ha2⟩
    have := forallynexynotinclosurex a (Ne.intro ha2)
    contradiction
```

```
    exact closure_eq_iff_isClosed.mp closureeqself
  push Not at existsdiffx
  have : {x} = Set.univ := by
    ext y
    constructor
    · simp
    simp [existsdiffx]
  exact closure_subset_iff_isClosed.mp fun □ a□  a_1 => existsdiffx a
```

```
=== Messages ===
[error] 27:14: Unknown identifier `helperforsingleton`
=== Infoview State (first 5 line(s)) ===
[info] 39:1: [infoview] tactic state:
no goals
```