Chain bounding, the leanest proof of Zorn's lemma, and an illustration of computerized proof formalization

Abstract. We present an exposition of the *Chain Bounding Lemma*, which is a common generalization of both Zorn's Lemma and the Bourbaki-Witt fixed point theorem. The proofs of these results through the use of Chain Bounding are amongst the simplest ones that we are aware of. As a by-product, we show that for every poset P and function f from the powerset of P into P, there exists a maximal well-ordered chain whose family of initial segments is appropriately closed under f.

We also provide an introduction to the process of "computer formalization" of mathematical proofs by using *proofs assistants*. As an illustration, we verify our main results with the Lean proof assistant.

1. INTRODUCTION This paper grew out of the search for an elementary proof of Zorn's Lemma. One such proof was obtained by the first author, which is similar to the one by Lewin [1].

After a careful examination, the authors realized that the method of proof actually yielded a pair of new, similar principles: *Chain Bounding* and the *Unbounded Chain Lemma*, which state the impossibility of finding strict upper bounds of linearly ordered subsets of posets. The first one is more fundamental, since it does not depend on the Axiom of Choice, which states:

(AC) For every family $\{A_i \mid i \in I\}$ of nonempty sets, there exists a function $f : I \to \bigcup_{i\in I} A_i$ such that for all $i \in I$, $f(i) \in A_i$.

When Chain Bounding is coupled with AC, it implies the second principle and then Zorn's Lemma. Chain Bounding also implies the Bourbaki-Witt fixed point theorem; all these results are in Section 3.

Our original proof of Chain Bounding proceeded by contradiction, where a few relevant concepts were defined; this proof is essentially the one that appears in Appendix A, where it is used to show Zorn's Lemma in a self-contained manner. We realized that it was better instead to present those concepts ("good chains" and their comparability) independently to obtain positive results. These appear in Section 2, and Chain Bounding is now proved as a consequence of its main Theorem 4, the existence of a greatest good chain. Nevertheless, the main advantage of Chain Bounding in comparison to Theorem 4 is its straightforward statement and consequences, which make it more appealing as a "quotable principle".

As a way to discuss the correctness and level of detail of our arguments, we introduce the subject of "computer formalization" of mathematics in Section 4, and in Appendix B we present a brief description of the verification of our main results, using the *Lean* proof assistant.

A word on our intended audience. As one of the reviewers indicated, different parts of this paper can be better suited for varied kinds of readers. In general, the paper should be accessible to mature undergraduates, but the main focus changes a bit across sections. Instructors can benefit from having the succinct proof of Zorn's Lemma from Appendix A, or directly adopting Chain Bounding as a tool if they are teaching a course on posets. The discussion from Corollary 8 to the end of Section 3 goes a little deeper into the details and assume a bit more of set-theoretic background. Finally, we hope that the general exposition of proof assistants be accessible to a wider public; the detailed example in Appendix B is a bit more challenging but might be attractive to younger students which are more familiar with computer technology.

2. THE GREATEST GOOD CHAIN We introduce some notation. Let P be a poset and $C \subseteq P$; we say that $s \in P$ is a *strict upper bound* of C if $\forall c \in C,\ c < s$. Furthermore, if $S \subseteq C$, we say that S is an *initial segment* of C ("$S \sqsubseteq C$") if for all $x \in C$, $x \leqslant y \in S$ implies $x \in S$. We will usually omit the word "initial" and simply say "S is a segment of C". The strict version of the segment relation is denoted by $S \sqsubset C$, that is, $S \sqsubseteq C$ and $S \neq C$. Finally, $\mathscr{P}(X)$ denotes the powerset of the set X.

Definition 1. Let $g : \mathscr{P}(P) \to \mathscr{P}(P)$ be given. We say that a chain $C \subseteq P$ is *good for* g if for all $S \sqsubset C$, $S \sqsubset g(S) \sqsubseteq C$.

When understood from the context, we omit "for g". We have the following key result.

Lemma 2 (Comparability). *Let P be a poset and $g : \mathscr{P}(P) \to \mathscr{P}(P)$. If C_1 and C_2 are good chains, one is a segment of the other.*

Proof. Let $\mathscr{S}$ be the family of mutual segments of both C_1 and C_2. Hence $\bigcup \mathscr{S}$ is also a mutual segment. If $\bigcup \mathscr{S}$ is different from both C_1 and C_2, then $g(\bigcup \mathscr{S})$ should be a mutual segment since both are good; but this contradicts the fact that $g(\bigcup \mathscr{S}) \nsubseteq \bigcup \mathscr{S}$. ∎

Lemma 3. *The union of a family $\mathscr{F}$ of good chains is a good chain.*

Proof. The union $U := \bigcup \mathscr{F}$ is a chain by Comparability.

Note that every good chain D such that $D \subseteq U$ is a segment of U: Suppose that $c \in U$ and $c < d \in D$. Then $c \in C$ for some good $C \in \mathscr{F}$. If C is a segment of D, we have $c \in D$ and are done. Otherwise, the converse relation holds by Comparability and then we also have $c \in D$.

We will see that U is good. Let $S \subsetneq U$ be a proper segment of U. Then, there exists $d \in U$ such that $\forall c \in S,\ c < d$. Let D be a good chain such that $d \in D$. Since D is a segment of U, all those c belong to D. We conclude $S \subseteq D$, and since S is a segment of U, it is a segment of D and it is proper because $d \in D \smallsetminus S$. Then $g(S)$ is segment of D, and hence $g(S)$ is a segment of U. ∎

By considering the union of *all* good chains, we readily obtain:

Theorem 4 (Greatest Good Chain). *Let P be a poset and $g : \mathscr{P}(P) \to \mathscr{P}(P)$. The family of all good chains has a maximum under inclusion.* ∎

3. CHAIN BOUNDING AND APPLICATIONS The following lemma is the key to all of what follows. In some sense, it might be regarded as a non-AC version of Zorn's lemma.

Lemma 5 (Chain Bounding). *Let P be a poset. There is no assignment of a strict upper bound to each chain in P.*

Proof. Assume, by way of contradiction, that $f(C)$ is a strict upper bound of C for each chain $C \subseteq P$. Hence C is a proper segment of $g(C) := C \cup \{f(C)\}$; extend this g arbitrarily to the rest of the subsets of P. By Theorem 4, there exists a greatest good chain U for g. But this is a contradiction, since $g(U)$ is easily seen to be a good chain, but $g(U) \not\subseteq U$. ■

The wording of the Chain Bounding Lemma is a bit awkward, since it is actually a negation. However, if we now invoke AC, we get the following more natural statement.

Lemma 6 (Unbounded Chain). *Assume AC. For every poset P there exists a chain $C \subseteq P$ with no strict upper bound.*

Proof. By way of contradiction, assume that for every chain $C \subseteq P$ there exists a strict upper bound. Using AC, let f assign to each C such a bound. But this contradicts Chain Bounding. ■

We now turn to applications. The first one is very simple proof of Zorn's Lemma (obviously taking into account the lemmas proved so far).

Corollary 7 (Zorn). *If a poset P contains an upper bound for each chain, it has a maximal element.*

Proof. By the Unbounded Chain Lemma, take $C \subseteq P$ without strict upper bounds. Then any upper bound of C must be maximal in P. ■

A direct, self-contained proof of Zorn's Lemma condensing all the ideas discussed up to this point appears in Appendix A below. This includes some simplifications that also apply to a direct proof of Chain Bounding; for instance, the definition of good chain is a bit shorter and one only needs to show that the union of all good chains is good.

Since Zorn's Lemma implies AC, we immediately have:

Corollary 8. *The Unbounded Chain Lemma is equivalent to AC over Zermelo-Fraenkel set theory.* ■

Kunen points out in [2] that, for those not familiar with set theory, it may not be clear why Zorn's Lemma should be true, since the best-known proofs make use of ordinals and transfinite recursion. Our highest hope is that after seeing our proof, the old joke turns into *"AC is obviously true, the Well-Ordering Theorem is obviously false, and Zorn's Lemma... holds by Chain Bounding!"*.

Lewin, in [1], provides a very short proof without the need for ordinals or recursion, but making use of well-ordered chains. In [3], Lang presents a proof using the Bourbaki-Witt fixed point theorem of order theory, without even employing the concept of well-ordered set. Finally, Brown [4] gives a beautiful and simple proof inspired by Lang's but without the need for Bourbaki-Witt. This proof is slightly indirect since it actually proves the Hausdorff Maximal Principle and considers "closed" subsets in the poset of chains of the original poset ordered by inclusion. (Other proofs in the same spirit can be found in Halmos [5] and Rudin [6].)

The proof we presented here is not as short as Lewin's but it is more elementary since there is no use of (the basic theory of) well-orders in an explicit way. The main difference in method that allows us to avoid them is the generalization of his definition of "conforming chains" by considering general initial segments instead of *principal* ones (i.e., of the form $\{x \in P \mid x \leqslant p\}$ for some $p \in P$). This move allows us to use the stronger expressiveness achieved by talking about general segments (indirectly referring to the powerset of P), but avoids referring to "second order" chains (i.e., chains in the poset of chains).

In spite of this simplification, the fundamental character of the concepts of well-order and well-foundedness in general should be strongly emphasized. These are unavoidable in a sense; actually good chains for functions that add at most one element (such as the one in the proof of Chain Bounding) are well-ordered:

Proposition 9. *Let $(P, <)$ be poset, $f : \mathscr{P}(P) \to P$ and let $g(C) := C \cup \{f(C)\}$. Every good chain for g is well-ordered by $<$.*

Proof. We leave to the reader the verification of the fact, under the assumptions, that every segment D of a good chain is good.

Let C be a good chain and assume $X \subseteq C$ is nonempty. Let S be the set of strict *lower* bounds of X in C. Since $X \neq \varnothing$, S is a proper segment of C, and goodness ensures that $S \neq g(S) = S \cup \{f(S)\}$ is also a segment of C. Since $f(S) \notin S$, there is some $x \in X$ such that $x \leqslant f(S)$. We claim that any such x must be equal to $f(S)$ and hence it is the minimum element of X. For this, consider the good subchain $D := \{c \in C \mid c \leqslant x\}$ of C. Since S is likewise a proper segment of D, $f(S) \in g(S) \subseteq D$ and hence we obtain the claim. ■

Moreover, it can be shown that a chain of a poset P is well-ordered if and only it is a good chain for some g as above.

Our second application is the aforementioned fixed point theorem.

Corollary 10 (Bourbaki-Witt). *Let P be a non-empty poset such every chain $C \subseteq P$ has a least upper bound. If $h : P \to P$ satisfies $x \leqslant h(x)$ for all $x \in P$, then h has a fixed point, i.e., there is some $x \in P$ such that $x = h(x)$.*

Proof. Assume by way of contradiction that $x < h(x)$ for all $x \in P$. But then $f(C) := h(\sup C)$ immediately contradicts Chain Bounding. ■

Note that the greatest good chain for $C \overset{g}{\longmapsto} C \cup \{h(\sup C)\}$ is the least complete subposet of P closed under h, and its ordinal length is (the successor of) the number of iterations of h needed to reach its least fixed point starting from the bottom element of P.

It is relevant here that Chain Bounding does not depend on AC, since Bourbaki-Witt can be proved without using it. This is another reason why we consider Chain Bounding the central item of this work.

4. COMPUTER FORMALIZATION OF MATHEMATICS Although Lewin's proof is even shorter than ours, it could be debatable whether it is more straightforward for a general audience. Specifically, some acquaintance with the basic theory of well-orders seems to be required to understand it in full; in other words, not all the details of the proof are completely disclosed (note, e.g. the Mathematics Stack Exchange questions [7] and [8] and their answers).

Our combo Chain-Bounding/Zorn is not immune to the same criticism: Some tricky details could still have been swept under the rug—or, even worse, some mistake. The question presents itself: How could one assert that all details of a proof have been taken care of, and how to be sure that absolutely all steps are correct? This concern will certainly connect in the reader's mind to the "rigorization" process in Mathematics during the late XIX century and perhaps to the foundational aspects of mathematical logic. It might also raise an alert of sorts (recalling Russell and Whitehead's *Principia*) of the infeasibility of the task.

The fact of the matter is that thanks to a steady development in Computer Science, modern programming technologies called *proof assistants* (PAs) have been created, that allow one to write a mathematical proof in a formal language and the computer can then verify it for correctness. This process is called "formalization", "(formal) verification", or "mechanization".

Use cases PAs not only make it possible to trace every single detail of simple results such as the ones in this paper; highly non trivial and recent research-level mathematical results have been checked in diverse PAs. Notable formalizations include the Four Color Theorem [9], the Odd-Order Theorem [10] (both using the Coq PA), and the proof of the Kepler Conjecture [11] (using a combination of HOL-Light and Isabelle/HOL PAs). Even more recently, Gowers, Green, Manners, and Tao settled the Polynomial Freiman-Ruzsa conjecture [12], sharing their results on November 9th, 2023; Tao started a project [13] to formalize them (including many preliminaries) using the *Lean* PA [14] and it was finished [15] less than a month after the manuscript was released! The process was described by Tao in his blog [16].

There are also many ongoing attempts to use PAs in teaching. A regular workshop [17] discusses the potential uses of the latter and other computer technologies in education, and there are actually many courses that already use PAs. Examples are Massot's calculus course [18] for the second semester of BSc, and Macbeth's Math 2001 at Fordham University, supported by an excellent introductory material [19] to Lean. More pointers to the use of Lean in education can be found in Avigad's talk [20] at the Fields Institute.

A bird's eye comparison To this day, there are plenty of fine pieces describing the different kinds PAs available, and their use, benefits and pitfalls. To name a few, Harrison, Urban, and Wiedijk recall the rich history of PAs in their [21], while Koutsoukou-Argyraki shares her experiences using Isabelle/HOL in [22] and, after presenting an insightful summary, interviews some of the relevant figures involved with this technology in her [23]. Even a recent article in Nature magazine discusses PAs [24]. Having all these resources at hand, we will only briefly overview some details of current PAs.

PAs come in a great variety, both internally/foundationally speaking and from the user point of view. Regarding foundations, most of them differ from the traditional set-theoretic ones, being based on several kinds of *type theories*. For the purpose of this paper, we can take "type" and "set" to be synonyms, but the main difference is that every mathematical object in consideration belongs to *exactly one* type; as an example of this stipulation, a natural number n is not to be considered a real one, but instead there is an appropriate embedding $\mathbb{N} \hookrightarrow \mathbb{R}$ which sends n to the corresponding real. Isabelle/HOL and HOL-Light are based on the "simple" theory of types; the axiomatics of the latter PA is surprisingly terse, and a description can be found at Hales [25]. On the other hand, Coq and Lean support "dependent type theory", which is stronger.[1]

There are notable differences also in the language used by PA to write proofs and the overall user interfaces. Some PAs are actually full-fledged programming languages

[1] There have been interesting debates (e.g., Buzzard's challenge to the Isabelle community [26]) on what is the actual impact of the differing strengths of PAs' foundations concerning the possibility to formalize research-level mathematics. We only point out that HOL (simple type theory) systems are weaker than Zermelo-Fraenkel set theory with Choice, which in turn is weaker than the dependent type theory with universes used in Lean. We are also not discussing the *constructivist* aspect of foundations, which may enable or prevent the extraction of algorithms from the formalized proofs.

that are also suited to formalized proofs (e.g., Agda, Lean), while some others may produce computer programs from a proof (Coq, Isabelle/HOL). Concerning the way proofs are written, two major flavors are available [21, Sect. 6.2]:

- *declarative* proofs state intermediate facts that lead to the expected result ("We have X, hence Y, and then Z"); and
- *procedural* ones, which consist in a series of instructions to be performed to current "goal" or assertion to be proved ("Subtract x from both sides, unfold f's definition, conclude").

In a very rough first approximation, procedural proofs are easier to write (and more useful for exploration) and declarative proofs are easier to read (in some cases, even without the need to use the corresponding PA to examine them).

Most PAs support a combination of both methods, but there are some extreme cases. For instance, the Naproche variant of Isabelle is completely declarative in the sense that it tries to solve each intermediate step automatically; this results in beautiful formal proofs [27] but it is somewhat limited in power. On the other end, proofs in Agda are constructive, and what one actually does is to define a function that provides the witness/result required by the statement.

Most of our experience comes from using both Isabelle and Lean. Isabelle allows its users to write *proof documents* that can be read without the need of a computer; but it should be emphasized that this requires extra effort, and people do not always formalize things in this way. Nevertheless, the HOL variant of Isabelle has a noteworthy component (*Sledgehammer*) able to deal automatically with many of the intermediate steps. Lean, on the other hand, uses a more procedural way to work. After consulting on this point, it was observed that human-readable versions of the proofs could be derived from the formal artifacts, as exemplified during Massot's talk [28] at the 2023 IPAM Workshop on *Machine Assisted Proofs*. Tools like Sledgehammer are also being actively developed for Lean.

A further difference between these two PAs might be the composition of the user base. Thanks to several high-profile projects, many mathematicians have started to use Lean during the recent years; and while there are mathematicians working with Isabelle, we believe their community hosts a majority of computer scientists.

In Appendix B, we present the formalization of the main results of the paper using the Lean PA, which may serve to showcase the look of the code using it and some characteristics of its user interface. We also invite the reader to visit the website

`https://leanprover-community.github.io/learn.html`

where many indications can be found on how to start using Lean, and the online forum

`https://leanprover.zulipchat.com/`

fosters the growing community of users and is very welcoming to newcomers.

A. ZORN'S ON ITS OWN For ease of reference, we streamline the arguments above to obtain a compact version of the proof.

Recall that for a poset P, $s \in P$ is a *strict upper bound* of $C \subseteq P$ if $\forall c \in C,\ c < s$; and that $S \subseteq C$ is a *segment* of C if for all $x \in C$, $x \leqslant y \in S$ implies $x \in S$.

Lemma (Zorn). *If a poset P contains an upper bound for each chain, it has a maximal element.*

Proof. Assume by way of contradiction that $(P, \leqslant)$ does not have a maximal element. Hence, for every chain $C \subseteq P$ there exists a strict upper bound (otherwise, any upper bound of C would be maximal). Using the Axiom of Choice, let g assign $C \cup \{s\}$ to each chain $C \subseteq P$, where s is any such bound.

A chain $C \subseteq P$ is deemed to be *good* whenever

(*) If $S \neq C$ is a segment of C, then $g(S)$ also is.

We need the following property of good chains:

(*Comparability*) If C_1, C_2 are good, one is a segment of the other.

To prove it, let $\mathscr{S}$ be the family of mutual segments of both C_1 and C_2. Hence $\bigcup \mathscr{S}$ is also a mutual segment. If $\bigcup \mathscr{S}$ is different from both C_1 and C_2, then $g(\bigcup \mathscr{S})$ should be a mutual segment by (*); but this contradicts the fact that $g(\bigcup \mathscr{S}) \not\subseteq \bigcup \mathscr{S}$.

Let U be the union of all good chains, which is a chain by Comparability.

Note that every good chain D is a segment of U: Suppose that $c \in U$ and $c < d \in D$. Then $c \in C$ for some good C. If C were not a segment of D, then the converse relation would hold by Comparability and then we would also have $c \in D$.

We will see that U is good. Let $S \subsetneq U$ be a proper segment of U. Then, there exists $d \in U$ such that $\forall c \in S,\ c < d$. Let D be a good chain such that $d \in D$. Since D is a segment of U, all those c belong to D. We conclude $S \subseteq D$, and since S is a segment of U, it is a segment of D and it is proper because $d \in D \setminus S$. Then $g(S)$ is segment of D, and hence $g(S)$ is a segment of U.

We reach a contradiction, since $g(U)$ is also a good chain, but $g(U) \not\subseteq U$. ■

B. A COMPUTER FORMALIZATION IN LEAN In this section, we present the formalization of the main results of the paper using the Lean proof assistant. It takes up 420 lines of code and it is available at

this link,

For the remainder of this section, we will focus on explaining only a fraction of the details involved (and in particular, some notations will not be dealt with); our main objective is to make a point that it is possible to translate mathematical reasoning into the computer, in a way that at least partially resembles the way it is done on paper. We hope that the reader's curiosity will be sufficiently motivated in order to visit the mentioned resources and to learn more about formalization and Lean.

We start our Lean file by *importing* basic results on chains, and the definition of complete partial orders (which appear in the Bourbaki-Witt Theorem).

```
import Mathlib.Order.Chain
import Mathlib.Order.CompletePartialOrder

variable {α : Type*}
```

The last line above indicates that we will be talking about a "type" α (which, in the type theory of Lean roughly corresponds to a set, or perhaps more appropriately, a set *underlying* some structure). Greek letters are commonly used for types, and here this α will replace our P from above.

We highlight some of the basic definitions. For instance, "S is a (proper) segment of C" is defined in the following way:

```
def IsSegment [LE α] (S C : Set α) : Prop := S ⊆ C ∧ ∀ c ∈ C, ∀ s ∈ S, c
    ⩽ s → c ∈ S

def IsPropSegment [LE α] (S C : Set α) : Prop := IsSegment S C ∧ S ≠ C
```

The arguments S and C appear declared as belonging to the powerset of α, which in Lean is written as `Set` α. The declaration in square brackets is an *implicit* argument stating that α belongs to the class of types having the $\leqslant$ notation defined (which is the bare minimum to be able to interpret the right hand side). After a few more lines, the declaration `variable [PartialOrder α]` states that we will be assuming a partial order structure on α.

We can actually set up infix notation for `IsSegment` and `IsPropSegment` in order to be able to write expressions as $S \sqsubseteq C$ and $S \sqsubset C$, as shown below.

After a new concept is introduced, a customary requisite is to write some extremely basic lemmas which allow one to work with it. These are referred to by the name *API*, an acronym for "application programming interface", a concept that comes from Computer Science. In our formalization, part of the API comprises all the possible transitivity lemmas involving $\sqsubseteq$, $\sqsubset$, or both.

We describe the formalization of the fact, used at the beginning of the proof of Lemma 2, that the union of a family of segments is a segment. The formalized statement is the following (where $\bigcup_0$ denotes the operator of union of a family), and the `by` keyword signals the start of the (tactic) proof:

```
lemma sUnion_of_IsSegment {F : Set (Set α)} (hF : ∀M ∈ F, M ⊑ C) : ⋃₀ F
    ⊑ C := by
```

Since $\bigcup_0 F \sqsubseteq C$ is defined by a conjunction, its justification is *constructed* by providing proofs for each conjunct. Each of those proofs appear indented and signaled by "·" below. We will analyze the first sub-proof line by line.

```
constructor
· intro s sInUnionF
  obtain ⟨M, MinF, sinM⟩ := sInUnionF
  exact (hF M MinF).1 sinM
· intro c cinC s sInUnionF cles
  obtain ⟨M, MinF, sinM⟩ := sInUnionF
  exact ⟨M, MinF, (hF M MinF).2 c cinC s sinM cles⟩
```

Right after writing `constructor` and the subsequent dot, the VS Code editor echoes:

```
α : Type u_1
inst † : PartialOrder α
C : Set α
F : Set (Set α)
hF : ∀ M ∈ F, M ⊑ C
⊢ ⋃₀ F ⊆ C
```

This "InfoView" lists all terms available to work (hypotheses are also included as "`Prop`ositional" terms), and the current *goal* (which, for this sub-proof, is the inclusion on the last line).

The natural way of producing a proof of that inclusion (defined by `∀ ⦃s⦄, s ∈ ⋃₀ F → s ∈ C`), is to *introduce* two new variables named `s` and `sInUnionF`,

```
· intro s sInUnionF
```

whose types ("α" and "`s ∈ ⋃₀ F`", respectively) are deduced from the previous goal. Now the InfoView turns into

```
α : Type u_1
inst † : PartialOrder α
C : Set α
F : Set (Set α)
```

```
  hF : ∀ M ∈ F, M ⊑ C
  s : α
  sInUnionF : s ∈ ⋃₀ F
  ⊢ s ∈ C
```

From `sInUnionF`, which states by definition that `s` belongs to some element of `F`, we obtain such an element `M` and further terms/hypothesis that state the relations among them with the next line (where the "`:=`" can be read as "from"):

```
   obtain ⟨M, MinF, sinM⟩ := sInUnionF
```

After this tactic, the propositional variables `MinF` and `sinM` state that `M ∈ F` and `s ∈ M`, respectively. Finally, we combine all the elements available by using some of the benefits of the type-theoretic framework:

- Logical constructs like implications and universal quantifiers behave as functions. For instance, the hypothesis `hF` (of type `∀ M, M ∈ F → M ⊑ C`) can be fed with the term `M` to obtain the implication `hf M` (having type `M ∈ F → M ⊑ C`) and the latter can be applied to `MinF` (a term for the antecedent) to obtain a term `hf M MinF` for the consequent `M ⊑ C`.
- The conjunction behaves as a Cartesian product, where components correspond to each conjunct. Hence the first component `(hf M MinF).1` is a term justifying `M ⊆ C` = `∀ ⦃s⦄, s ∈ M → s ∈ C`.

By applying the last term obtained to `sinM`, we obtain `exact`ly what we were looking for, and the sub-proof ends.

```
    exact (hF M MinF).1 sinM
```

For the definition of goodness, we declare our own class consisting of types supporting a partial order, and we add an otherwise unspecified g. A special type of comment (a *docstring*) describes the concept introduced:

```
/--
A partial order with an *expander* function from subsets to subsets. In
    main applications, the
expander actually returns a bigger subset.
-/
class OrderExpander (α : Type*) [PartialOrder α] where
  g : Set α → Set α
```

Assuming the appropriate structures on α we are finally able to write down the definition.

```
variable [PartialOrder α] [OrderExpander α]

def Good (C : Set α) := IsChain ( · ≤ ·) C ∧ ∀ {S}, S ⊏ C → S ⊏ g S ∧ g
    S ⊑ C
```

A further class concerns partial orders with an "f",

```
class OrderSelector (α : Type*) [PartialOrder α] where
  f : Set α → α
```

and a statement that each type supporting it is an "instance" of `OrderExpander` in a canonical way. This is justified by presenting $C \overset{g}{\longmapsto} C \cup \{f(C)\}$ as the witness.

```
instance [PartialOrder α] [OrderSelector α] : OrderExpander α := ⟨fun C =>
    C ∪ {OrderSelector.f C}⟩
```

We skip directly to the statement of the Unbounded Chain Lemma,

```
lemma unbounded_chain [PartialOrder α] [Inhabited α] :
    ∃ C, IsChain ( · ⩽ ·) C ∧ ¬ ∃ sb : α, ∀ a ∈ C, a < sb
```

which moreover assumes α to be nonempty (more precisely, that has a designated element) for simplicity, and the proof of Zorn's Lemma using it:

```
lemma zorn [PartialOrder α] [Inhabited α]
    (ind : ∀ (C : Set α), IsChain ( · ⩽ ·) C → ∃ ub, ∀ a ∈ C, a ⩽ ub) : ∃
    (x : α), IsMaximal x := by
  obtain ⟨C, chain, subd⟩ := unbounded_chain (α := α)
  push_neg at subd
  obtain ⟨ub, hub⟩ := ind C chain
  existsi ub
  intro z hz
  by_contra zneub
  obtain ⟨a, ainC, anltz⟩ := subd z
  exact anltz $ lt_of_le_of_lt (hub a ainC) $ lt_of_le_of_ne' hz zneub
```

We comment briefly on some of the tactics employed in this elementary proof. As before, `obtain` decomposes the statement of `unbounded_chain`, and in particular `subd` is a term asserting the truth of `¬ ∃ sb : α, ∀ a ∈ C, a < sb`. The tactic `push_neg` applies the De Morgan rules transforming it into `∀ (sb : α), ∃ a ∈ C, ¬a < sb`. The obtained upper bound `ub` for the (strictly) unbounded chain is presented as a witness to the existential quantifier of the conclusion by using the `existsi` tactic. After introducing variables `h` and `hz`, the `by_contra` tactic starts a proof by contradiction where the new hypothesis (the negation of the goal appearing immediately before) is stored in the variable `zneub`.

We would like to add a final word on the description of this proof of Zorn's Lemma as the "leanest" one: Mathlib's version of its formalization [29] was *ported* (translated) from Isabelle/HOL. This new proof was originally conceived for and written in Lean.

Acknowledgment We are very grateful to both reviewers and to the Editorial Board for their comments that helped to enhance our presentation. Particularly, we thank Reviewer 3 for an extremely dedicated reading, and Reviewer 2 for insightful comments on the intended audience of the paper.

References

[1] Lewin J. A Simple Proof of Zorn's Lemma. The American Mathematical Monthly. 1991;98(4):353-4.
[2] Kunen K. Set Theory. 2nd ed. Studies in Logic. College Publications; 2011. Revised edition, 2013.
[3] Lang S. Real and Functional Analysis. Graduate Texts in Mathematics. Springer New York, NY; 2012.
[4] Brown K. Zorn's Lemma; 2010. https://pi.math.cornell.edu/~kbrown/6310/zorn.pdf. Mathematics 6310.
[5] Halmos PR. Naive Set Theory. Undergraduate Texts in Mathematics. Springer; 1960. Available from: https://books.google.com.ar/books?id=x6cZBQ9qtgoC.
[6] Rudin W. Real and Complex Analysis. 3rd ed. Series in Higher Mathematics. McGraw-Hill; 1987.

[7] WillG. Question on a step of A Simple Proof of Zorn's Lemma by Lewin; 2020. https://math.stackexchange.com/q/3616209(version:2020-04-28). Mathematics Stack Exchange.

[8] Harumi C. Proof of Zorn's lemma clarification; 2019. https://math.stackexchange.com/q/3269781 (version: 2019-06-21). Mathematics Stack Exchange.

[9] Gonthier G. The Four Colour Theorem: Engineering of a Formal Proof. In: Kapur D, editor. Computer Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg; 2008. p. 333-3.

[10] Gonthier G, Asperti A, Avigad J, Bertot Y, Cohen C, Garillot F, et al. A Machine-Checked Proof of the Odd Order Theorem. In: Proceedings of the 4th International Conference on Interactive Theorem Proving. ITP'13. Berlin, Heidelberg: Springer-Verlag; 2013. p. 163-79. Available from: https://doi.org/10.1007/978-3-642-39634-2_14.

[11] Hales T, Adams M, Bauer G, Dang TD, Harrison J, Hoang LT, et al. A Formal proof of the Kepler conjecture. Forum Math Pi. 2017;5:e2, 29.

[12] Gowers WT, Green B, Manners F, Tao T. On a conjecture of Marton. arXiv e-prints. 2023 Nov:arXiv:2311.05762. Available from: https://arxiv.org/abs/2311.05762.

[13] A digitisation of the proof of the Polynomial Freiman-Ruzsa Conjecture in Lean 4; 2023. Repository available at https://teorth.github.io/pfr/.

[14] de Moura L, Ullrich S. The Lean 4 Theorem Prover and Programming Language. In: Platzer A, Sutcliffe G, editors. Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. vol. 12699 of Lecture Notes in Computer Science. Springer; 2021. p. 625-35. Available from: https://doi.org/10.1007/978-3-030-79876-5_37.

[15] Tao T. Polynomial Freiman-Ruzsa conjecture: Project Completed!; 2023. Announcement available at https://leanprover.zulipchat.com/#narrow/stream/412902-Polynomial-Freiman-Ruzsa-conjecture/topic/Project.20completed!.20.20Thoughts.20and.20reflections.3F/near/405996596.

[16] Tao T. Formalizing the proof of PFR in Lean4 using Blueprint: a short tour; 2023. Blog post available at https://terrytao.wordpress.com/2023/11/18/formalizing-the-proof-of-pfr-in-lean4-using-blueprint-a-short-tour/.

[17] Narboux J, Neuper W, Quaresma P, editors. Proceedings 12th International Workshop on Theorem proving components for Educational software. vol. 400 of Electronic Proceedings in Theoretical Computer Science. EPTCS; 2024. Available from: https://cgi.cse.unsw.edu.au/~eptcs/content.cgi?thedu23.

[18] Massot P. Notes de cours; 2023. Available at https://www.imo.universite-paris-saclay.fr/~patrick.massot/mdd154.

[19] Macbeth H. The Mechanics of Proof; 2023. Available at https://hrmacbeth.github.io/math2001/.

[20] Avigad J. Formal Assistants and Mathematics Education; 2022. Video: https://youtu.be/rBii0Hws1ug?si=ohFC_RmTsyaqsgUv&t=3294. Available from: https://www.andrew.cmu.edu/user/avigad/Talks/fields_education.pdf.

[21] Harrison J, Urban J, Wiedijk F. History of Interactive Theorem Proving. In: Siekmann JH, editor. Computational Logic. vol. 9 of Handbook of the History

of Logic. Elsevier; 2014. p. 135-214. Available from: https://doi.org/10.1016/B978-0-444-51624-4.50004-6.

[22] Koutsoukou-Argyraki A. Formalising Mathematics—in Praxis; A Mathematician's First Experiences with Isabelle/HOL and the Why and How of Getting Started. Jahresbericht der Deutschen Mathematiker-Vereinigung. 2020 Oct;123(1):3-26. Available from: http://dx.doi.org/10.1365/s13291-020-00221-1.

[23] Koutsoukou-Argyraki A. What can formal systems do for mathematics? A discussion through the lens of proof assistants. In: Löwe B, Sarikaya D, editors. 60 Jahre DVMLG (Special Issue for the 60th Anniversary of the DVMLG). vol. 48 of Tributes. London: College Publications; 2022. p. 139-69.

[24] Castelvecchi D. Mathematicians welcome computer-assisted proof in 'grand unification' theory. Nature. 2021;595:18-9.

[25] Hales TC. Formal proof. Notices Amer Math Soc. 2008;55(11):1370-80.

[26] Buzzard K. The rise of formalism in mathematics; 2022. ICM2022 plenary talk. Available from: https://www.youtube.com/live/SEID4XYFN7o?si=uv1hPyHigtiMtttA&t=3578.

[27] De Lon A, Koepke P, Lorenzen A, Marti A, Schütz M, Sturzenhecker E. Beautiful Formalizations in Isabelle/Naproche. In: Kamareddine F, Sacerdoti Coen C, editors. Intelligent Computer Mathematics: 14th International Conference, CICM 2021, Timisoara, Romania, July 26–31, 2021, Proceedings. Cham: Springer International Publishing; 2021. p. 19-31.

[28] Massot P. Formal mathematics for mathematicians and mathematics students; 2023. Talk at the UCLA Institute for Pure & Applied Mathematics. Available from: https://www.ipam.ucla.edu/abstract/?tid=17912&pcode=MAP2023.

[29] Mathlib proof of Zorn's lemma; 2024. Available at https://leanprover-community.github.io/mathlib4_docs/Mathlib/Order/Zorn.html#Notes.