Teaching "Foundations of Mathematics" with the LEAN Theorem Prover

Mattia Luciano Bottoni

Supervisors:
Prof. Dr. Alberto Cattaneo
Dr. Elif Saçikara
Marius Furter

Master's Thesis

Institue of Mathematics
University of Zurich
Spring Semester 2024

Abstract

This thesis aims to understand if the theorem prover LEAN positively influences students' understanding of mathematical proving. To this end, we perform a pilot study concerning freshmen students at the University of Zurich (UZH). While doing so, we apply certain teaching methods and gather data from the volunteer students enrolled in the "Foundations of Mathematics" course. After eleven weeks of study covering some exercise questions implemented with LEAN, we measure LEAN students' performances in proving mathematical statements, compared to other students who are not engaged with LEAN. For this measurement, we interview five LEAN and four Non-LEAN students and we analyze the scores of all students in the final exam. Finally, we check significance by performing a t-test for independent samples and the Mann-Whitney U-test.

Acknowledgments

First and foremost, I would like to thank Prof. Dr. Alberto Cattaneo for giving me the opportunity to write the Master's thesis, Dr. Elif Saçikara for always supporting my work and being there for me, even when she herself was crowded with work, I could not have done this thesis without her, and Marius Furter for joining my thesis by the end, giving me highly valued inputs and supporting me as if he had been my supervisor himself.

The work presented here would also not have been possible without the students of the "Foundations of Mathematics" course who agreed to be interviewed by me. For privacy reasons, I will thank them anonymously. A special thank you goes to the five volunteer students who came to my meetings. You guys gave me a lot of your free time voluntarily and I appreciate it very much. My biggest thanks go to Aron Ebinger, I really enjoyed discussing LEAN and linear algebra with you and I hope that our collaboration was as fruitful for you as it was for me. At this point, I would also like to express my gratitude towards Dr. Hao Wu, the lecturer of the "Foundations of Mathematics" course in the Fall semester of 2023, for being so open in letting me work with his students and for all the opportunities he gave me regarding my thesis.

Although I did not have much personal contact with the following people, the LEAN community, the mathematics department of UZH and UZH itself have removed so many boulders along my way. I thank you and may my gratitude reach you all and remove boulders from your daily lives too.

Next I would like to thank my parents, Fulvio and Laura Bottoni-Morelli. No words can describe the love I feel towards them. Without them, I would not even exist and I would not have made it this far in life. Thank you for always putting us children first and for all the love and support you have given us. Thank you, Paolo and Sofia for always being at my side since the days I can remember. Growing up with you all has made me the man I am today. Thank you Balu and Chili († 2023) just for existing, you two have so often lowered my stress level with your calming presence.

No one can survive maths and physics without friends to support you and share the pain with you. I am lucky to have met some really good people during my Master's studies. Thank you, Mikael, Joel, Oswaldo, Irena, Yannick, Mathäus and the other Irchel Pirates, for spending so much time laughing and struggling with me over the past four years. I salute you and dedicate part of my work to you. From the physics side, I would like to thank Matthias, Alina, Yuri, Philipp and Tanja for helping me survive the other half of my degree. I could not have asked for a nicer, funnier and smarter learning group.

I am happy to say that I also had a life besides my studies. These years have been made so much better by the good people around me. A special shout-out to Elia, Larissa, Nikola, Noa and Arjuna. Thank you for bringing a bit of Graubünden with you to Zurich. My time here would not have been the same without you and I value your friendship.

Speaking of Graubünden, thank you Linard Bardill for contributing so much to my childhood with your songs and stories. Rediscovering you last year saved me from an impending burnout, by soothing my soul with your music.

My last thank you goes to a very important person. Flurina, you came into my life after I had just finished my Bachelor's degree and started the most advanced academic mountain I would climb. Having you by my side gave me perspective and made me feel like that mountain was almost a small hill. Thank you for being a part of my life and for brightening up every day I wake up next to you. Am a tai...

“§$qqqqqqqqqqqqqqqqqqqqqqq3e$” (Balu, 07.02.2024)

Contents

1 Introduction

Automated theorem provers and proof assistants have been around since the 1960s, when the first formal language AUTOMATH was developed [Geu09]. Since then, many theorem provers have come forth, such as *Agda*, *Isabelle* and *Coq*. Each of them has its own strengths and weaknesses. For example, *Agda* has no proof automation and *Isabelle* has no dependent types in its underlying structure, meaning that predicate logic can only be done in a challenging way [Pau86; AV24]. While automated theorem provers are more vulnerable to mistakes, proof assistants are robust, but usually very difficult to use.

To take advantage of both systems, LEAN was introduced in 2013 by Leonardo de Moura at Microsoft Research Redmond, and is one of the programming languages that acts as a bridge between proof assistant and automated theorem prover [Avi+23]. It differs from *Coq*, for instance, LEAN uses a more mathematician-friendly syntax and is supported by a large community of mathematicians [Com18], while *Coq* is still used more in computer sciences circles [AV19; Com24; Com23b; AV23a]. Still, both are based on adaptations of the calculus of constructions.

The calculus of constructions is formed by two main topics: type theory and λ-calculus. Type theory was developed by Bertrand Russell around 1908 to solve paradoxes such as the Russell paradox by introducing a *hierarchy of types* and by not allowing types of the *same order* to refer to themselves [Rus08]. Type theory has been developed further since then, but set theory remains the foundation most taught in universities.

Independently, Alonzo Church came up with the idea of λ-calculus in the 1930s. It is a concept that allows us to abstract functions in such a way that we can implement them in computers [Rod22]. When we add type theory to the λ-calculus, we initially get a system that is *Turing complete*, but not restrictive enough to do mathematical logic with. By adding restrictions and building new structures for this system, we end up with the calculus of constructions. With this constructive foundation, one can do first-order logic on a computer and verify proofs.

Even though LEAN is based on such a high level of abstract mathematics, it has gained much popularity in teaching undergraduate students in recent years, and even studies about the effects of teaching mathematics with LEAN are coming up [Avi19; TI21]. While trends show that teaching with LEAN could have positive learning effects, it remains a challenge to integrate teaching LEAN into current university curricula.

This thesis aims to provide further evidence of the positive effects of teaching with LEAN concerning freshman students' performance in the "Foundations of Mathematics" course taught at the Institute of Mathematics of the University of Zurich. To this end, we learn LEAN ourselves by implementing seven LEAN exercise sheets based on the content of the course and we organise eleven sessions over one semester in which volunteer students are taught the foundations of mathematics with LEAN.

To be able to carry out such a study, we first consider certain teaching methods like writing appealing goals for the students [Mag78], trying to turn extrinsic motivation into intrinsic motivation [DR93], varying a lot between frontal teaching and experimented learning [Wah13], choosing an appropriate difficulty level [VC78], differentiating teaching for different skill levels [Wod14], embracing digital teaching resources [Pet14] and creating a learn-efficient class climate [CMP14].

Secondly, we consider the data gathering. More explicitly, we follow the structure of Thoma and Iannone by conducting interviews with LEAN and Non-LEAN students to compare their proof writing and by checking whether LEAN students perform better in the final exam [TI21]. To measure the significance of the results, we perform two statistical tests, the t-test for independent samples and the Mann-Whitney U-test [DAT24c; DAT24a]. We additionally use a questionnaire with open and closed questions to better understand the motivation and contentment of the LEAN students. It is important to mention that the names of the students considered are all anonymised using Romansh language starting with **L** for LEAN students and **N** for Non-LEAN students. The gender of the names does not represent the gender of the actual students.

The structure of the thesis is as follows. In Section 2, we provide the necessary background for the notions in the foundations of mathematics concerning the readers at the bachelor level. We further introduce the necessary background on type theory and λ-calculus as simply as possible, without affecting the core ideas. Section 3 discusses the teaching methods considered for the meetings with the volunteer students and explains how we collect our data. The results, supported by sample solutions of students and statistical tests, are then presented in Section 4. In particular, we present an evaluation of the students' proof structure, their exam scores, and their progress. Finally, we discuss our experiences learning and teaching (with) LEAN in Section 5.

2 Preliminaries

Here we talk about the topics the students learn in the course “Foundations of Mathematics” at UZH in the Fall semester of 2023 that are implemented and discussed in LEAN. We also talk about LEAN’s logical foundations and the way LEAN works.

2.1 Foundations of Mathematics

The course “Foundations of Mathematics” is a first-semester course at UZH, where the students learn about logic and proofs, how sets and functions are defined and categorized and how one can build up the natural numbers in an axiomatic way using set theory. In the Fall semester of 2023, almost the whole course is based on a book by Hammack [Ham18]. Only one of the last topics about natural numbers relies on a different text by Stewart [ST15]. At the end of the course, the lecturer introduces order relations, once again based on the latter reference. In the next few subchapters, relating to these two references, we present the most important results from each topic. Only the results we need for the interview questions are explained here.

Propositional Logic

Propositional logic is the study of statements and logical connections. It gives rise to the set-theoretic foundations of mathematics, as it gives us the possibility to derive the truth value of a given statement from known statements by combining them correctly.

Definition 2.1.1. A **statement** is a sentence or mathematical expression that is either definitely true or definitely false.

Example 2.1.2. Here are some (non-)examples of statements.

- In summer, the weather is generally warmer.
- We have that $2 < 5$.
- “add two to both sides of the equation” is not a statement.

Statements can be connected in certain ways. Depending on the truth value of those statements, the truth value of their combination may vary. We can give an overview of the corresponding truth values in so-called *truth tables.*

Definition 2.1.3. The following **truth table** shows the truth values for the logical connectives $\wedge$ (*and*), $\vee$ (*or*), $\neg$ (*not*), $\rightarrow$ (*implies*) and $\leftrightarrow$ (*equivalent to*), where P and Q are both statements.

P	**Q**	**¬P**	**P ∧ Q**	**P ∨ Q**	**P → Q**	**Q → P**	**P ↔ Q**
T	T	F	T	T	T	T	T
T	F	F	F	T	F	T	F
F	T	T	F	T	T	F	F
F	F	T	F	F	T	T	T

Table 1: Truth table for simple logical connectives.

One can use more than one logical connective to combine statements. It is then possible that we create two statements that "say the same thing", even though they look different.

Definition 2.1.4. Two statements are called **logically equivalent** if they have the same truth value under all possible assignments of truth values to the statements occuring in them.

The following is an example for two such logically equivalent statements. The yellow marked truth values are the same for all cases.

Example 2.1.5. The statements $P \rightarrow Q$ and $(\neg Q) \rightarrow (\neg P)$ are logically equivalent since they have the same truth values.

Proof. We compare the truth values of the two statements in a truth table.

P	Q	$\neg P$	$\neg Q$	$P \rightarrow Q$	$(\neg Q) \rightarrow (\neg P)$
T	T	F	F	T	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

Table 2: Logical equivalence proved by a truth table.

$\square$

Remark 2.1.6. We denote two logically equivalent statements P and Q as follows $P = Q$.

The so-called *deMorgan laws* are two examples of logical equivalence.

Theorem 2.1.7 (deMorgan laws)**.** *Let P and Q be statements. Then the following are logical equivalences*

- $\neg(P \wedge Q) = (\neg P) \vee (\neg Q)$,
- $\neg(P \vee Q) = (\neg P) \wedge (\neg Q)$.

Proof. We can again prove this using truth tables.

P	Q	$\neg P$	$\neg Q$	$P \wedge Q$	$\neg(P \wedge Q)$	$(\neg P) \vee (\neg Q)$
T	T	F	F	T	F	F
T	F	F	T	F	T	T
F	T	T	F	F	T	T
F	F	T	T	F	T	T

Table 3: One of the deMorgan laws proved by a truth table.

P	Q	$\neg P$	$\neg Q$	$P \vee Q$	$\neg(P \vee Q)$	$(\neg P) \wedge (\neg Q)$
T	T	F	F	T	F	F
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	F	T	T

Table 4: The second deMorgan law proved by a truth table.

$\square$

Remark 2.1.8. In the interviews we ask the students to prove the deMorgan laws differently, i.e. without a truth table.

In addition to the logical connectives, we have two so-called quantifiers. They tell us if a statement holds for at least a single or for all “things”.

Definition 2.1.9. The symbols $\exists$ and $\forall$ are called **quantifiers**.

- The symbol $\exists$ stands for “there exists” or “there is”.
- The symbol $\forall$ stands for “for all” or “for every”.

We show both quantifiers used in a single example.

Example 2.1.10. Consider the expression $\forall x \in \mathbb{R},\ \exists\, c \in \mathbb{R},\ c \cdot x = 0$. This means that *for all* real numbers x, *there exists* a real number c such that $c \cdot x = 0$ holds.

Basic Set Theory

In this topic, we give a better understanding of the “things” mentioned above. We start by defining a set.

Definition 2.1.11. A **set** is a collection of things. This things are called the **elements** of the set.

Let us see some explicit examples.

Example 2.1.12. The following are all examples of sets:

- $S_1 := \{1, 2, 3, 4\}$,
- $S_2 := \{car, yellow, sky\}$,
- $S_3 := \{x \mid x \text{ is even}\}$,
- $S_4 := \mathbb{N}$ (or $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$).

As one can see, the elements of a set can take on all shapes and sizes. In our context, however, elements are mostly numbers. Independent of the type of these elements, we can always say how many elements we have in a set.

Definition 2.1.13. The **cardinality** of a set S is the number of elements in that set. We denote the cardinality of S with $|S|$ or $\#S$.

Relating to the example above, let us see some cases of different cardinality.

Example 2.1.14. The sets S_1 and S_2 from before have cardinality $|S_1| = 4$ and $|S_2| = 3$ respectively.

Not all sets need to have a finite number of elements. We can define when a set is finite or infinite as follows.

Definition 2.1.15. We say that the cardinality of a set S is **finite** if there exists a natural number n such that $|S| = n$. If $|S| > n$ holds for all natural numbers, we say that S is an **infinite set**.

Similar to the logical connectives, we want to have a notion for when two sets are equal.

Definition 2.1.16. Two sets are said to be **equal** if they contain exactly the same elements.

The order of the elements is not relevant, as one can see in the example below.

Example 2.1.17. The sets $\{1, 2, 3, 4\}$ and $\{4, 3, 2, 1\}$ are equal.

We are now at a point where we can explore more sophisticated set structures like the ones below.

Definition 2.1.18. There are three "special" sets to consider:

- Empty set: The **empty set** $\emptyset$ is the set with no elements.
- Subset: Let A and B be sets. If every element of B is also an element of A, then B is called a **subset** of A. We denote this by $B \subseteq A$.
- Power set: The set of all subsets of a set A is called the **power set** and it is denoted by $\mathcal{P}(A)$.

We present an explicit example for the power set. With this example, it should also become clear what subsets and the empty set are.

Example 2.1.19. Let $A := \{1, 2, 3\}$, then

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, A\}$$

Like with statements, there are certain combinations we can do to create new sets from given sets. These combinations are called operations.

Definition 2.1.20. We can define various operations with two sets A and B both lying in a universal set S:

- The **union** of A and B is the set: $A \cup B := \{x \mid x \in A \vee x \in B\}$.
- The **intersection** of A and B is the set: $A \cap B := \{x \mid x \in A \wedge x \in B\}$.
- The **difference** of A and B is the set: $A \setminus B := \{x \mid x \in A \wedge x \notin B\}$.
- The **complement** of either A or B is the set: $\overline{A} := \{x \mid x \notin A\}$.
- The **cartesian product** of A and B is the set: $A \times B := \{(a, b) \mid a \in A \wedge b \in B\}$.

We can use *Venn diagrams* to illustrate sets informally. In a Venn diagram, sets are usually drawn as circles or ovals, while the universal set is a rectangle around the sets.

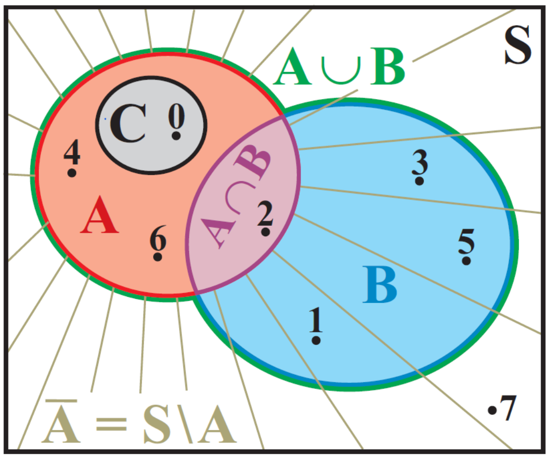


Figure 1: Venn diagram [Wet19]

We only have 26 alphabetical symbols, so if we want to have a huge number of sets to combine, it is convenient to use indexed sets.

Definition 2.1.21. We can label sets with a subscript, for example i. The sets $A_1, A_2, A_3, \ldots$ or simply $A_i,\ i \in I$ are called **indexed sets**. The symbol I denotes the set of the indexes.

Remark 2.1.22. We often use $I = \mathbb{N}$, but I can be any kind of set.

Using these indexed sets, we can define bigger unions and intersections of sets.

Definition 2.1.23. Let A_i be indexed sets, where $i \in I$.

- The **union** of all A_i is the set: $\bigcup_{i \in I} A_i := \{x \mid \exists\ i \in I, x \in A_i\}$.
- The **intersection** of all A_i is the set: $\bigcap_{i \in I} A_i := \{x \mid \forall\ i \in I, x \in A_i\}$.

We have a look at two examples.

Example 2.1.24.

- Let $I = \{1, 2, 3\}$ and $A_1 = \{1\}, A_2 = \{1, 2\}, A_3 = \{1, 2, 3\}$. Then,
$$\bigcup_{i \in I} A_i = \{1, 2, 3\} \text{ and } \bigcap_{i \in I} A_i = \{1\}.$$
- Now let $J = \mathbb{N}$ and $A_j = \{x \mid x \text{ divides } j\}$. Then,
$$\bigcup_{j \in J} A_j = \mathbb{N} \text{ and } \bigcap_{j \in J} A_j = \{1\}.$$

Proofs

There are certain procedures one can follow when conducting a proof. The most relevant ones are discussed here. We follow each definition with a proof example.

Definition 2.1.25. Direct proofs are the most intuitive ones, they start with the sentence "suppose P" and end with the sentence "therefore we have Q". In between those two, we try to get from one to the other using logical statements, definitions and mathematical facts.

In [Ham18], it is said that while one can write each transformation or step during the proof on a single line at the beginning, we should write a single paragraph at a later point. In LEAN, we have exactly this line-by-line style of proof, as can be seen in Chapter 2.2), which helps one see the small steps one should take to get to the desired result. This is why we also use this step-by-step form in the examples.

Example 2.1.26. We want to prove that if we add two natural even numbers a and b (x is even if there exists $n \in \mathbb{N}$ such that $x = 2 \cdot n$), the resulting number will be even again. The proof goes as follows.

Proof.
Suppose that a and b are even.
Then $a = 2 \cdot n$ and $b = 2 \cdot m$ for some $n, m \in \mathbb{N}$.
Let $c = a + b$.
Then $c = 2 \cdot n + 2 \cdot m$. This means that $c = 2 \cdot (n + m)$, where $(n + m) \in \mathbb{N}$.
Therefore, c is even.
That concludes our proof. $\square$

While we could prove conditional statements only with direct proofs, it is often convenient to use the logical equivalence of $P \Rightarrow Q$ and $(\neg Q) \Rightarrow (\neg P)$ and prove a statement the other way around. We proved this logical equivalence in the Subsection 2.1.

Definition 2.1.27. Proofs starting with "Assume that Q does not hold." and ending with "Therefore P does not hold.", are called **contrapositive proof**.

Example 2.1.28. We want to prove that for a natural number n, the statement "n^2 is even" implies that "n is even" using a contrapositive proof. The proof looks like this.

Proof.
Suppose that n is not even (i.e. $n = 2 \cdot c + 1$, for some $c \in \mathbb{N}$).
Then, $n^2 = n \cdot n = (2 \cdot c + 1) \cdot (2 \cdot c + 1)$.
But this gives us $n^2 = 4 \cdot c^2 + 4 \cdot c + 1 = 2 \cdot (2 \cdot c \cdot (c + 1)) + 1$.
And $(2 \cdot c \cdot (c + 1)) \in \mathbb{N}$.
Therefore, n^2 is not even. $\square$

There is a third important proof structure, which is often confused with the contrapositive proof by undergraduate students.

Definition 2.1.29. When doing a **proof by contradiction**, we assume that the statement is false and then do a direct proof until we end up in a contradiction.

Example 2.1.30. We prove the same statement as above but with a proof by contradiction. In this case, the proof goes like this.

Proof.
Assume (wrongly) that n^2 is even but n is not.
This means that $n^2 = 2 \cdot c$ and $n = 2 \cdot k + 1$ for some $c, k \in \mathbb{N}$.
We have that $n^2 = n \cdot n = (2 \cdot k + 1) \cdot (2 \cdot k + 1)$.
Which leads to $2 \cdot c = 4 \cdot k^2 + 4 \cdot k + 1$.
We rewrite this as $1 = 2 \cdot (c - 2 \cdot k^2 - 2 \cdot k)$.
Therefore, 1 is even.
But this is a contradiction. Which concludes that the original statement is true. $\square$

Carefully compare the examples given for the contrapositive proof and the proof by contradiction and understand the fundamental differences between those two.

Mathematical Induction

When we want to prove that a statement holds for all natural numbers, we can do a proof by induction.

Definition 2.1.31. Let $S_1, S_2, \ldots$ all be mathematical statements we want to prove to be true. In a **proof by induction**, we prove this as follows.

- Prove that the first statement S_1 is true (base case).
- Assume that the statement S_n is true for some n (induction hypothesis).
- Prove that S_{n+1} holds (induction step).

One can then conclude that all statements S_k are true.

Let us see how we can use a proof by induction to prove a statement.

Example 2.1.32. We can prove the famous Gauss' formula, which apparently, he found when he was nine, using induction. We want to prove that $1+2+3+...+k = (k \cdot (k+1))/2$. We start with the base case:
Assume that $k = 1$. Then we have that $1 = (1 \cdot (1+1))/2 = 1$.
This concludes the base case.
Assume that that $1 + 2 + 3 + ... + n = (n \cdot (n+1))/2$ holds for some $n \in \mathbb{N}$.

Our goal is to prove that $1 + 2 + 3 + ... + n + (n + 1) = ((n + 1) \cdot ((n + 1) + 1))/2$.
We can rewrite the left-hand side using the induction hypothesis as
$(n \cdot (n + 1))/2 + (n + 1)$.
But this is equal to $(n \cdot (n + 1))/2 + (2 \cdot (n + 1))/2 = (n^2 + 3n + 2)/2$.
Which is the same as $((n+1)\cdot((n+1)+1))/2 = (n^2+2n+1+n+1)/2 = (n^2+3n+2)/2$
By induction we have now proved Gauss' formula.

Relations and Functions

Using the notion of a subset, we can define all kinds of relations between elements.

Definition 2.1.33. A **relation** on a set A is a subset $R \subseteq A \times A$. We write $(x, y) \in R$ or xRy.

Some relations have special properties that can be very useful.

Definition 2.1.34. A relation R can have the following characteristics:

- A relation is called **reflexive** if $xRx, \forall x \in A$.
- A relation is called **symmetric** if $xRy \Rightarrow yRx, \forall x, y \in A$.
- A relation is called **transitive** if $xRy \wedge yRz \Rightarrow xRz, \forall x, y, z \in A$.

Definition 2.1.35. If a relation R is reflexive, symmetric and transitive, we say that R is an **equivalence relation**.

We already know some relations that are equivalence relations and some that are not.

Example 2.1.36. The relation $=$ is an equivalence relation. The relations $<$ and $\leq$ however are not, since they are both not symmetric and $<$ is also not reflexive.

Similar to relations, we can define functions.

Definition 2.1.37. Suppose A and B are sets. A **function** f from A to B (denoted $f : A \to B$) is a relation $f \subseteq A \times B$, satisfying the property that for each $a \in A$ the relation f contains exactly one ordered pair of the form (a, b). The statement $(a, b) \in f$ is abbreviated as $f(a) = b$.

A function is a relation between two different sets. Similar to relations, a function can have special characteristics.

Definition 2.1.38. Let f be a function from A to B. Then,

- f is called **injective** if $f(x) = f(y) \Rightarrow x = y, \forall x, y \in A$.
- f is called **surjective** if $\forall b \in B, \exists a \in A$ with $b = f(a)$.
- f is called **bijective** if it is injective and surjective.

The sets A and B on which f is defined on play a role for whether f is *injective*, *surjective* or *bijective*. There is an example of this below.

Example 2.1.39. The function $f : \mathbb{R} \to \mathbb{R},\ f(x) = x^2$ is neither injective nor surjective, as $f(x) = f(-x)$ and as we never reach negative numbers.
The function $f : \mathbb{R}_+ \to \mathbb{R}_+,\ f(x) = x^2$ however is both injective and surjective and is therefore a bijection.

A bijective function is the counterpart to an equivalence relation. We will see that having a bijective function between two sets can be very powerful.

Cardinality

Earlier we have seen what it means for two sets to be equal. Now we would like to talk about when two sets have equal cardinality. It can be shown that having equal cardinality is an equivalence relation.

Definition 2.1.40. Two sets A and B (not necessarily finite) are said to have **equal cardinality**, written $|A| = |B|$, if there exists a bijective function $f \colon A \to B$. If no such function exists, then A and B are said to have **unequal cardinality**.

Let us see a bijection proving that $\mathbb{N}$ and $\mathbb{Z}$ have equal cardinality.

Example 2.1.41. The sets $\mathbb{N}$ and $\mathbb{Z}$ have equal cardinality, as the function

$$f \colon \mathbb{N} \longrightarrow \mathbb{Z},\ f(n) = \begin{cases} \frac{n}{2}, & \text{for } n \text{ even,} \\ -\frac{n+1}{2}, & \text{for } n \text{ odd} \end{cases}$$

is a bijection.

We still have to talk about different kinds of infinite sets. It turns out that not all infinities are the same.

Definition 2.1.42. Let A be a set. Then A is called **countably infinite** if there exists a bijection $f \colon \mathbb{N} \to A$. A is said to be **countable** if A is finite or countably infinite. A is called **uncountable** if A is infinite and $|\mathbb{N}| \neq |A|$ i.e., if there exists no bijection $f \colon \mathbb{N} \to A$.

Contrary to the set $\mathbb{Z}$, the reals do not have the same cardinality as the set $\mathbb{N}$.

Example 2.1.43. The real numbers $\mathbb{R}$ are uncountably infinite.

We see that using functions we can compare the cardinality of two sets.

Definition 2.1.44. Let A and B be sets. Then,

- $|A| = |B|$ means that there is a bijection $A \to B$.
- $|A| < |B|$ means that there is an injection $A \to B$, but no bijection $A \to B$.
- $|A| \leq |B|$ means that there is an injection $A \to B$.

The following example explains how we can argue about cardinalities like that. The proof is left for the reader.

Example 2.1.45. $A < \mathcal{P}(A)$, since there only exists a injection from A to $\mathcal{P}(A)$.

One can also prove equal cardinality, without finding a bijection between two sets.

Theorem 2.1.46 (Cantor-Bernstein-Schröder)**.** *Let A and B be any sets. If there exist injections $f\colon A \to B$ and $g\colon B \to A$, then there exists a bijection $h\colon A \to B$.*

Remark 2.1.47. This is called the Cantor-Bernstein-Schröder theorem. It basically says that $(|A| \leq |B|) \wedge (|B| \leq |A|) \Rightarrow (|A| = |B|)$.

Example 2.1.48. With the Cantor-Bernstein-Schröder theorem, we can prove that $\mathbb{R}$ and $\mathcal{P}(\mathbb{N})$ have equal cardinality.

Natural Numbers

We already talked about natural numbers in the previous chapters. They are the most intuitive numbers, namely the ones we use to count. One way to define the natural numbers is explained below.

Definition 2.1.49. The **Peano axioms** are three axioms that describe the natural numbers. Together with the **existence axiom**, they give us our known set $\mathbb{N}$. Suppose that there exists a set $\mathbb{N}$ and a function $succ\colon \mathbb{N} \to \mathbb{N}$ such that

- (P1): the function *succ* is not surjective. There exists $0 \in \mathbb{N}$ such that $succ(n) \neq 0$ for all $n \in \mathbb{N}$.
- (P2): the function *succ* is injective.
- (P3): if $S \subseteq \mathbb{N}$ is such that $0 \in S$ and $n \in S \Rightarrow succ(n) \in S$ for all $n \in \mathbb{N}$, then $S = \mathbb{N}$.

- (existence): There exists a set $\mathbb{N}$ and a function $succ\colon \mathbb{N} \to \mathbb{N}$ satisfying the above three axioms.

While we have seen the induction principle and the definition of natural numbers, we still need to define recursive functions, to be able to define addition, multiplication and so on.

Theorem 2.1.50. *If X is a set, $f\colon X \to X$ a function, and $c \in X$, then there exists a unique function $\phi\colon \mathbb{N} \to X$ such that*

(i) $\phi(0) = c$.

(ii) $\phi(s(n)) = f(\phi(n)), \forall n \in \mathbb{N}$.

The proof of the recursion theorem is rather technical and will not be considered in this thesis. Readers interested in the proof can find it in [ST15].

With the recursion theorem, we can now define three so-called recursive functions.

Definition 2.1.51. We mostly use the following three recursive functions from $\mathbb{N}$ to $\mathbb{N}$.

- (**Addition**) The function $(n + m) : \mathbb{N} \to \mathbb{N}$ defined by:
  - ·) $n + 0 = n$ for all $n \in \mathbb{N}$,
  - ·) $n + succ(m) = succ(n + m)$ for all $n, m \in \mathbb{N}$.
- (**Multiplication**) The function $(n \cdot m) : \mathbb{N} \to \mathbb{N}$ defined by:
  - ·) $n \cdot 0 = 0$ for all $n \in \mathbb{N}$,
  - ·) $n \cdot succ(m) = n \cdot m + n$ for all $n, m \in \mathbb{N}$.
- (**Power**) The function $(n^m) : \mathbb{N} \to \mathbb{N}$ defined by:
  - ·) $n^0 = 1$ for all $n \in \mathbb{N}$,
  - ·) $n^{succ(m)} = n^m \cdot n$ for all $n, m \in \mathbb{N}$.

Order Relations

Being an equivalence relation is rather strict. We can define different types of order relations. Order relations can define an ordering on a set. Contrary to equivalence relations, they do not have the notion of symmetry, as we want to define “smaller” and “bigger” elements.

Definition 2.1.52. A relation R is called a **partial order** if

- aRa (reflexivity),
- $(aRb \wedge bRc) \Rightarrow aRc$ (transitivity),
- $(aRb \wedge bRa) \Rightarrow a = b$ (anti-symmetry).

A partial order can only relate to some of the elements of the sets. If we want an order relation relating to all elements, we need to ask for totality. Below one can explicitly see what we mean.

Example 2.1.53. The relation $a \mid b$ on the on the set $\mathbb{N} \setminus \{0\}$ is a partial order, since not all $a, b \in \mathbb{N}$ satisfy this relation, e.g. $3 \nmid 5$.

Definition 2.1.54. A relation R is called a **weak order** if

- $(aRb \wedge bRc) \Rightarrow aRc$ (transitivity),
- $(aRb \wedge bRa) \Rightarrow a = b$ (anti-symmetry),
- $aRb \vee bRa$ (totality).

There is yet a third order relation we consider.

Definition 2.1.55. A relation S is called a **strict order** if

- $(aSb \wedge bSc) \Rightarrow aSc$ (transitivity),
- $(aSb \ \veebar \ bSa \ \veebar \ a = b)$ (trichotomy),

where $\veebar$ stands for exclusive or, i.e. only one of the three statements can be true at once.

The two relations we saw before that were not equivalence relations can be shown to be order relations.

Example 2.1.56. The most trivial examples of weak and strict order are $\leq$ and $<$ respectively. One can check that these two relations satisfy all required characteristics.

There is a specific question one can ask when deciding if we have a weak or a strict order. Just take an element a from the set and check whether aRa i.e. whether $(a, a) \in R$ or $a = a$, i.e. $(a, a) \notin R$.

Example 2.1.57. If we consider $<$ and $\leq$ on the natural numbers, we see that $5 \not< 5$ but $5 \leq 5$. So $\leq$ is indeed a weak order and $<$ a strict order.

2.2 LEAN

The LEAN project was launched by Leonardo de Moura at Microsoft Research Redmond in 2013 [Avi+23]. It is a proof assistant written mainly in C++ and in LEAN itself (its own programming language). With proof assistants, one can usually describe the way computers help with proof writing in two ways: *automated theorem proving* and *interactive theorem proving.* While the first helps one "find" a certain aspect of a proof, the latter focuses on verifying if a given proof is correct. LEAN aims to be the bridge between those two functions [Avi+23]. LEAN is based on the calculus of constructions with inductive types. In the following, we explain this foundation.

Type Theory and the Way to the Calculus of Constructions

Most foundational courses, as well as the one the students visited, build the foundations of mathematics on set theory, a theory that studies sets as mathematical objects and contains a list of axioms to conduct mathematical logic [FBL73]. While it is a very common foundation, it has its weaknesses and it is certainly not the only one. This subsection describes the way the foundations in LEAN are implemented and it is mostly based on [Rod22], [Ret22] and [Pro13]. We talk about type theory and λ-calculus to make our way up to a LEAN-specific type of calculus of constructions. Since the complexity of these theories goes beyond the scope of this thesis, we leave it to the readers to consult the references given for more details, they are highly recommended for interested readers.

Type Theory

Bertrand Russell developed type theory around 1908, at a time when mathematicians were investigating the paradoxes of set theory like the Russell paradox which states that: if $R = \{x \mid x \notin x\}$, then $R \in R$ if and only if $R \notin R$, which is a contradiction. Russell was able to solve this by introducing the hierarchy of types and stating that a statement in the same order of types could not refer to itself [Rus08]. So in the example of Russell's paradox, R would be of the same order type as the sets $\{x \mid x \notin x\}$ and therefore one could not substitute x by R. We will see how this *type theory* fundamentally betters computational mathematics soon.

Untyped λ-calculus

A few years after Russell developed type theory, Alonzo Church came up with the idea of λ-calculus. We first explain the untyped λ-calculus and state why it is useful for computation. Only after that, will we see how powerful it becomes when combined with type theory. For now, we refer to λ-calculus when talking about untyped λ-calculus. The idea of λ-calculus is to write functions in an abstract form. A very simple example would be the function $f(x) = x + 1$, which is denoted as $\lambda x, x + 1$ in λ-calculus. This is called a λ-term. Usually, we denote a λ-term with $\lambda x.$ and not λx, but since λ-calculus is written with a comma in LEAN, we also use this notation. We can now define the set of all λ-terms inductively as:

$$\Lambda = V \mid (\lambda V, \Lambda) \mid (\Lambda\Lambda)$$

where V denotes a set of variable symbols. We usually write x, y or z for variables and M, N for λ-terms. This special notation of the set Λ means that the set Λ is built inductively using the following three rules:

- (Variable) If $x \in V$, then $x \in \Lambda$.
- (Application) If $M, N \in \Lambda$, then $(MN) \in \Lambda$.
- (Abstraction) If $x \in V$ and $M \in \Lambda$, then $(\lambda x, M \in \Lambda)$.

The following example of some λ-terms explains what we mean.

Example 2.2.1. Let $x, y \in V$. Then,

- $x, y \in \Lambda$ (by the variable rule),
- $(xy) \in \Lambda$ (by the application rule),
- $(\lambda x, (xy)) \in \Lambda$ (by the abstraction rule),
- $((\lambda x, (xy))(\lambda x, (xy))) \in \Lambda$ (again by the application rule).

We have a special identity in this setting. The terms M and N are said to be *syntactically equal*, denoted by $M \equiv N$, if they represent the same λ-term. Note that the terms $(\lambda x, (xy))$ and $(\lambda z, (zy))$ are not syntactically equal, so our notion of equality is quite restrictive. We can overcome this by introducing *α-conversion*, but before we do this, we have to give a new definition, namely the notion of *free* and *bound* variables. Intuitively one can say that the bound variables are like the variables we know for example from functions, and the free variables can be thought of as constants.

Definition 2.2.2 (Set of free variables)**.** The **set of free variables** of a λ-terms is defined recursively as follows.

- (Variable) $FV(x) = \{x\}$ for every $x \in V$.
- (Abstraction) $FV(\lambda x, M) = FV(M) \setminus \{x\}$ for every $x \in V$ and $M \in \Lambda$.
- (Application) $FV(MN) = FV(M) \cup FV(N)$ for every $M, N \in \Lambda$.
- We say $M \in \Lambda$ is **closed** if $FV(M) = \emptyset$.

Remark 2.2.3. The *bound* variables are then the non-free variables. For example in the abstraction rule, x would be a bound variable.

With this, we are now able to define *α-conversion*:

Definition 2.2.4 (α-conversion)**.** Let $M \in \Lambda$, let $M^{x \to y}$ denote the λ-term in which each free occurrence of x in M has been replaced by y. We define **α-conversion**, which we denote by $=_\alpha$, as the smallest equivalence relation over Λ in which the following conditions hold:

- (Renaming) $\lambda x, M =_\alpha \lambda y, M^{x \to y}$ if y is neither a free nor a bound variable in M.
- (Compatibility) If $M =_\alpha N$, then $ML =_\alpha NL$, $LM =_\alpha LN$ and $\lambda z, M =_\alpha \lambda z, N$, for every $L \in \Lambda, z \in V$.

Remark 2.2.5. The condition in the renaming part basically tells us that one cannot substitute a free variable with another letter that looks the same as a bound variable in the expression. For example, we cannot do this substitution:

$$\int (x + c)\, dx \neq_\alpha \int (c + c)\, dc$$

We are now able to change variable names in λ-terms and let them still keep their meaning. What we would like to do next is to see how to substitute values for variables in λ-terms, i.e. how to apply a term to an expression.

Definition 2.2.6 (Substitution)**.** Let $M, N \in \Lambda$ and $x \in V$. We define $M[x := N]$ (to be read as M in which x has been substituted by N) inductively as follows.

- (Variable) $x[x := N] \equiv N$ and $y[x := N] \equiv y$ if $x \not\equiv y$.
- (Abstraction) $(\lambda y, P)[x := N] \equiv \lambda z, (P^{y \to z}[x := N])$ where $z \in V \setminus FV(N)$, for every $P \in \Lambda$.

- (Application) $(PQ)[x := N] \equiv (P[x := N])(Q[x := N])$, for every $P, Q \in \Lambda$.

The reason we rename the variable y to z in the λ-term P for the abstraction rule is to avoid a clash with occurrences of y in N.

We now introduce the notion of *one-step β-reduction.* This gives us a way to evaluate lambda expressions.

Definition 2.2.7 (One-step β-reduction)**.** We define **one-step β-reduction**, denoted as $\to_\beta$, as follows.

- (Reduction) $(\lambda x, M)N \to_\beta M[x := N]$, for every $M, N \in \Lambda, x \in V$.
- (Compatibility) If $M \to_\beta N$, then $ML \to_\beta NL$, $LM \to_\beta LN$ and $\lambda z, M \to_\beta \lambda z, N$ for every $L \in \Lambda, z \in V$.

An explicit example helps us to understand how one-step β-reduction works.

Example 2.2.8. β-reduction just describes how we apply a "function" on "something". For example, the function $\lambda x, x + x$ applied to the number 2 would be β-reduced like this: $(\lambda x, x + x)(2) \to_\beta (x + x)[x := 2] = 2 + 2$ $(= 4)$. The second rule that comes with β-reduction just tells us in which contexts we can still apply it.

Remark 2.2.9. A term of the form $(\lambda x, M)N$ is called *redex* and the reduced term $M[x := N]$ is called *contractum.*

The problem with one-step β-reduction is that it is not transitive. For example, while $(\lambda xy, x + y)(z)(z) \to_\beta (\lambda y, z + y)(z)$ and $(\lambda y, z + y)(z) \to_\beta z + z$, we do not have $(\lambda xy, x + y)(z)(z) \to_\beta z + z$. We introduce reduction paths to deal with this.

Definition 2.2.10 (Reduction path)**.** Let $M \in \Lambda$.

- A **finite reduction path** from M is a finite sequence of λ-terms $N_0, N_1, \ldots, N_n$ such that $N_0 \equiv M$ and $N_i \to_\beta N_{i+1}$ for every $0 \leq i \leq n$.
- An **infinite reduction path** from M is an infinite sequence of λ-terms $N_0, N_1, \ldots$ such that $N_0 \equiv M$ and $N_i \to_\beta N_{i+1}$ for every $i \in \mathbb{N}$.

We arrived at a point where we can define an equivalence relation for β-reduction.

Definition 2.2.11 (β-reduction, β-conversion)**.** Let $M, N \in \Lambda$. We write $M \twoheadrightarrow_\beta N$ if there exists a finite reduction path $M \equiv N_0, \ldots, N_n \equiv N$. We call this relation **β-reduction**.

We define **β-conversion** as the smallest equivalence relation containing $\twoheadrightarrow_\beta$ and denote it by $=_\beta$.

Remark 2.2.12. We treat $M =_\beta N$ as two equivalent, but not equal terms. This is also important during computation. For example, the terms $(\lambda x, x+2)(1)$ and $(\lambda x, x-1)(4)$ are equivalent by β-conversion, since they both reduce to 3, but they are not equal and cannot be β-reduced into each other.

There is a name for fully reduced β-expressions, they are called β-normal forms:

Definition 2.2.13 (β-normal form, β-normalising)**.** Let $M \in \Lambda$.

- We say that M is in **β-normal form** if M does not contain any redex.
- We say that M is **β-normalising** or that M has a β-normal form if there exists $N \in \Lambda$ in β-normal form such that $M =_\beta N$. Such N is the β-normal form of M

It can be shown that this β-normal form is unique. This follows directly from the Church-Rosser theorem.

Theorem 2.2.14 (Church-Rosser)**.** *Let $M, N_1, N_2 \in \Lambda$ such that $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow N_2$. Then, there exists $N_3 \in \Lambda$ such that $N_1 \twoheadrightarrow_\beta N_3$ and $N_2 \twoheadrightarrow_\beta N_3$.*

Remark 2.2.15. We do not discuss this proof in this thesis. Our reference, [Rod22] also forwards its readers to another source for the proof.

There remains one question: are all λ-terms β-normalising? For untyped λ-calculus, the answer is no. For example, we can consider the term $\Omega := (\lambda x, xx)(\lambda x, xx)$. We see that the only possible β-reduction is $\Omega \rightarrow_\beta \Omega$, which still has a redex. We can do this infinitely, so we never find a β-normal form for Ω. We also want to consider terms like $(\lambda u, v)(\Omega)$. If we reduce this to $(\lambda u, v)(\Omega) \rightarrow_\beta v$, we have found a β-normal form since v has no redex. If we try to reduce on Ω, we get the infinite reduction path

$$(\lambda u, v)(\Omega) \rightarrow_\beta (\lambda u, v)(\Omega) \rightarrow_\beta \dots$$

which does not have a β-normal form. This calls for new definitions:

Definition 2.2.16 (Weakly normalising, strongly normalising)**.** Let $M \in \Lambda$.

- We say that M is **weakly normalising** if there exists $N \in \Lambda$ in β-normal form such that $M \twoheadrightarrow_\beta N$.
- We say that M is **strongly normalising** if there do not exist any infinite reduction paths from M.

Remark 2.2.17. The λ-term $(\lambda u, v)(\Omega)$ is only weakly normalising but not strongly normalising, since while there exists a finite reduction path to a β-normal form, there also exists an infinite reduction path that never reaches a β-normal form. An example for a strongly normalising λ-term would be $(\lambda u, v)(x) \rightarrow_\beta v$. β-normalising is an umbrella term that contains weak and strong normalisation. Every weakly and strongly normalising term is also β-normalising. But knowing that a term is β-normalising does not tell us anything about whether it is weakly or strongly normalising.

Let us briefly recap what we did exactly and why. We introduced *α-conversion* to be able to change the name of bound variables. For that, we had to define *free* and *bound* variables of terms. Then we wanted to apply terms on other terms the way we apply functions to terms. We learned about *one-step β-reduction* to do this. But since this was not enough to define an equivalence relation, we introduced *reduction paths* and finally *β-conversion.* This allowed us to bring some terms into the *β-normal form*, a term expression where there are no more redexes. It turned out that not every term is *β-normalising.* We deal with this issue later.

The untyped λ-calculus was developed by Church to check which functions were computable using an algorithm. It turned out that Church computability (or λ-defineability), i.e. if a function could be written as a λ-term, was equivalent to Turing computability, which became to be known as the *Church-Turing thesis.* It turns out that the (untyped) λ-calculus is even Turing complete. However, there are some discrepancies. For example, terms like (xx) or (MM), as seen in the examples above, do not make much sense and there are terms which do not have a β-normal form. Adding type theory to the λ-calculus can overcome these problems. In the next section, we introduce the notion of *simple types* and step-by-step we arrive at a typed λ-calculus system which we denote as $\lambda\!\rightarrow$.

Simply typed λ-calculus

We start with the definition of simple types.

Definition 2.2.18 (Simple types)**.** Let $\mathbb{V} = \alpha, \beta, \gamma, \ldots$ be an infinite set of **type variables**. The set $\mathbb{T}$ of all **simple types** is defined as

$$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \rightarrow \mathbb{T}).$$

Remark 2.2.19. The set $\mathbb{T}$ therefore consists of types α and recursively constructed arrow types as $\alpha \rightarrow \beta$ or $\alpha \rightarrow (\alpha \rightarrow \beta)$.

To combine the simple types with the λ-terms, we need some new vocabulary.

Definition 2.2.20 (Statement, declaration, context, judgement)**.** The following definitions are used to distinguish objects in type theory.

- A **statement** is of the form $M : \sigma$, where $M \in \Lambda$ and $\sigma \in \mathbb{T}$. In such a statement, we call M the subject (or term) and σ the type.
- A **declaration** is a statement, where the subject is a variable, i.e. $x : \sigma$ for some $x \in V$.
- A **context** is a list of declarations. If we have no declarations, we call this the **empty context**, $\emptyset$.
- A **judgement** is of the form $\Gamma \vdash M : \sigma$, where Γ is a context and $M : \sigma$ is a statement.

We see that there are some terms, for which we cannot define their type properly, like the term (xx). It is therefore crucial that we do an extra step before defining the simply typed lambda calculus. We define the set of the so-called *pre-typed λ-terms.*

Definition 2.2.21 (Pre-typed λ-terms)**.** The set of **pre-typed λ-terms** is defined recursively as follows.

$$\Lambda_{\mathbb{T}} = V \mid (\Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}}) \mid (\lambda V : \mathbb{T}, \Lambda_{\mathbb{T}}).$$

Remark 2.2.22. Compare this definition to the definition of the untyped lambda calculus.

The expression (xx) is still a pre-typed λ-term, but we now begin to restrict this by introducing some *derivation rules.*

Definition 2.2.23 (Derivation rules in $\lambda\!\rightarrow$)**.** We have three derivation rules that restrict the terms we allow.

$$(var) \text{ If } x : \sigma \in \Gamma, \text{ then } \Gamma \vdash x : \sigma,$$

$$(appl) \,\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$(abst) \,\frac{\Gamma; x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma, M : \sigma \to \tau}$$

These rules are written in *sequent calculus*, a style of formal logic that is well known for the $\vdash$ (turnstyle) symbol. We do not talk about sequent calculus in more detail, we just want to mention that the turnstyle symbol can naively be read as “implies”. For example, the application rule says that given a context Γ which “implies” that

$M : \sigma \to \tau$ and $N : \sigma$ hold, then we can derive that under the context Γ, MN is of type τ. We actually read sequent calculus from below to the top when we do a derivation. So here we would argue that for $MN : \tau$ to hold under the context of Γ, we need to prove that under this context, the other two statements hold too.

We use these derivation rules for the following definition.

Definition 2.2.24 (Legal terms)**.** A pre-typed term M is called **legal** if there exists a context Γ and a simple type σ such that $\Gamma \vdash M : \sigma$, i.e. if we can find a derivation, such that this holds.

Legal terms have the following special property.

Theorem 2.2.25 (Strong normalisation)**.** *Every legal term M is strongly normalising.*

Our system $\lambda\!\!\to$ consists of all pre-typed terms that are legal. We started with simple types, but they did not all make sense. So we introduced the notion of pre-typed λ-terms and stated that a term M is legal if it has a derivation leading to $\Gamma \vdash M : \sigma$. We then only allowed the legal terms in the (simply) typed λ-calculus ($\lambda\!\!\to$). This solved the two problems we had in the untyped λ-calculus mentioned above, as terms like (xx) and (MM) are never legal and since all legal terms M are strongly normalising by the strong normalisation theorem.

There are in general three types of problems that one would like to solve in $\lambda\!\!\to$:

- (*Well-typedness* or *Typability*) $? \vdash \text{term} : ?$
- (*Type Checking*) $\text{context} \overset{?}{\vdash} \text{term} \ : \text{type}$
- (*Term Finding*) $\text{context} \vdash \ ? : \text{type}$

Well-typedness is simply the procedure to check whether a term is legal or not. We see that a term is given and we are asking for a context and a type. In the next case, *Type Checking*, we have a context and a term and would like to check whether a term has a given type in a given context, i.e whether we can find a valid derivation, under which the term has the given type. Last but not least, *Term Finding* gives us a context and a type, and we would like to find a term that given the context has the given type. Keep especially this last problem in mind.

One can show that all three problems are decidable in $\lambda\!\!\to$, i.e. there exists a general algorithm to solve them. If $\lambda\!\!\to$ would be strong enough to implement all the foundations of mathematics, we could build theorem provers that can solve any mathematical problem. However, this is not the case as we will see later. In more complex systems, *Well-typedness* and *Type Checking* remain decidable, and automated theorem provers like LEAN solve these problems all the time and give feedback on whether the terms are legal or not and if the typing matches. The last problem, *Term Finding*, is left for the mathematician and corresponds to finding the proof of a theorem, as we see right now.

One of the greatest changes when using type theory instead of set theory, is that the notion of sets and propositions are not two different things anymore. With that, a logic (a set of derivation rules) is always inherently integrated into the theory itself, unlike in set theory where logic must be defined separately. In the logic of $\lambda{\to}$, we can represent propositions as types and proof as terms of that type. Like with sets, where a term a is of type A when we say that $a \in A$, we can say that the term p is of type P when p is a proof of the proposition P. This is called the *PAT-interpretation*, which is short for proposition as types or proofs as terms. We now have a look at the proof that $A \Rightarrow B \Rightarrow B$ is true using the derivation rules of $\lambda{\to}$.

Example 2.2.26. The proof of $A \Rightarrow B \Rightarrow B$ in sequent calculus:

$$\dfrac{\dfrac{(1)\quad a:A; b:B \vdash b:B \text{ (var)}}{(2)\quad a:A \vdash \lambda b:B, b:B \to B}\,(abst)}{(3)\quad \emptyset \vdash \lambda a:A, \lambda b:B, b:A \to B \to B}\,(abst)$$

We are now able to implement proofs of propositions using λ-calculus! However, we do not stop here. The system $\lambda{\to}$ is too restrictive. Since all terms in $\lambda{\to}$ are strongly normalising, we lose Turing completeness, i.e. we cannot represent all Turing computable functions. On a more concrete level, since we have no self-application, we cannot define recursive functions or the factorial function. To solve these problems, we can build the so-called λ-cube. It consists of extensions of $\lambda{\to}$, which ultimately lead to a powerful system called the *calculus of constructions* (or CoC, λC for short), which is again Turing complete and less restrictive. In this thesis, we just mention the extensions and talk about their advantages and disadvantages. For more details on each extension, we recommend again the sources stated for this chapter.

λ2, $\lambda\underline{\omega}$ and λP

When we take the identity function in $\lambda{\to}$, it looks like this:

$$\lambda x : \alpha, x : \alpha \to \alpha.$$

This definition is only valid for x of type α. So if we would like to have the identity function for some $x : \beta$, we would need to construct a new function. In the system $\lambda 2$, we add the type of all types, denoted as $*$. With this, we can write the identity function as

$$\lambda \alpha : *, \lambda x : \alpha, x.$$

where we can insert any type $\alpha, \beta, (\alpha \to \alpha)$ or $(\alpha \to \beta)$ of type $*$. The problem arises when trying to find the type of such terms with variable types. The type $* \to \alpha \to \alpha$ is not equal to $* \to \beta \to \beta$. But since we want uniqueness of types for terms, this is a problem. We solve it by introducing Π-types. One can write $\Pi\alpha : *, \alpha \to \alpha$. Then we have that $(\lambda\alpha : *, \lambda x : \alpha, x) : \Pi\alpha : *, \alpha \to \alpha$, where the type α can be substitute by any type in $*$. With this, we have added polymorphisms to our system and as one can see for the term of the identity function, we have introduced terms that depend on types. One should keep that in mind. In the $\lambda 2$ system, we can now do *primitive recursion*, but we still have a strongly normalising system that is not Turing complete. So we need to further expand our system.

Let us investigate type structures next. Types like $\alpha \to \alpha$, $\beta \to \beta$ or even $(\alpha \to \beta) \to (\alpha \to \beta)$ all have the same structure of **Type** $\to$ **Type**, where the type on both sides is the same. We can generalize this structure as $\lambda\alpha : *, \alpha \to \alpha$. Since this is not a type itself, but it builds types, we call this a *type constructor*. The type of this type constructor would be $* \to *$. In order not to confuse types with types of types, we call types of the form $*, (* \to *), (* \to * \to *)$ and so on *kinds*. The set of kinds $\mathbb{K}$ consists of $\alpha : *$ (type constructors) and $(\lambda\beta : *, \beta : * \to *)$ (proper constructors). The type of all kinds is denoted as $\diamond$. From now on we use the letter s as a metavariable for either $*$ (types) or $\diamond$ (type constructors). In this system, called $\lambda\underline{\omega}$ (lambda weak omega), we were able to abstract more complex types, however, we do not have the notion of Π-types here. We later start to combine systems to get as many notions and abstractions as possible. But before we do so, we have a look at a system that introduces types depending on terms.

By now we have seen that we can build terms depending on terms in $\lambda{\to}$, terms depending on types in $\lambda 2$, with the general identity function, and types depending on types as type constructors in $\lambda\underline{\omega}$. For predicate logic, it is crucial that we can define types depending on terms. We have already talked about the PAT interpretation of typed λ-calculus. If we have a predicate proposition $P(x)$ that depends on an element x, we want to be able to build a type of the form $P(x)$, i.e. a type P depending on a term x, since finding a term is equivalent to finding the proof of the proposition. A proof p of type $P(x)$ (still denoted as $p : P(x)$) may then only exist for certain values of x. We can do all that in the system called λP, which introduces the notion of *dependent types*, while it again contains Π-types, we do not have proper kinds (proper type constructors) in this system. It is time we start combining the three systems to get better results.

We have found three systems that can help us do the following things:

- With $\lambda 2$, we can introduce the universal quantifier for types and we can build terms that depend on the type given.
- With $\lambda\underline{\omega}$, we can build more complex type structures, so-called (proper) type-constructors, which are types that depend on types. However, we do not have universal quantifiers in this system.
- With λP, we can make types depending on terms. This is very important to prove propositions that depend on a variable. In that system, we have the universal quantifier for terms, i.e. we can show that a proposition $P(x)$ (type) holds for all x (term).

It would be convenient if we were able to combine these three systems to get a system that inherits all of their characteristics. Fortunately, this is possible, and it can be illustrated by the λ-cube.

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda C \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda P2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda P\underline{\omega} \arrow[uu] \\
\lambda\to \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda P \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Figure 2: λ-cube (or Barendregt cube) [Rod22]

We can see that we can build all three combinations out of two of the three expansions of $\lambda\!\to$. The system in the up-right-back corner is the combination of all three systems. The so-called calculus of constructions (λC for short). The formal system of LEAN is an adaption of this system. This is why we want to look at it carefully.

Calculus of Constructions

To not confuse ourselves with the notion of types and terms, we introduce the notion of *expressions* for the system λC. Expressions are defined recursively as:

$$\mathcal{E} = V \mid * \mid \diamond \mid (\mathcal{E}\mathcal{E}) \mid (\lambda V : \mathcal{E}, \mathcal{E}) \mid (\Pi V : \mathcal{E}, \mathcal{E}).$$

We have the following derivation rules for λC:

$$(sort) \quad \emptyset \vdash * : \diamond$$

$$(var)\ \frac{\Gamma \vdash A : s}{\Gamma; x : A \vdash x : A}\ , \text{ if } x \notin \Gamma.$$

$$(weak)\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma; x : C \vdash A : B}\ , \text{ if } x \notin \Gamma.$$

$$(form)\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A, B : s_2}\ , \text{ where } B \text{ may depend on } x.$$

$$(appl)\ \frac{\Gamma \vdash M : \Pi x : A, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$(abst)\ \frac{\Gamma; x : A \vdash b : B \qquad \Gamma \vdash \Pi x : A, B : s}{\Gamma \vdash \lambda x : A, b : \Pi x : A, B}$$

$$(conv)\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}\ , \text{ if } B =_\beta B'.$$

Let us start explaining the first and easiest rule. The *sort* rule simply states that, even without any context, we have that a kind always has type $\diamond$. The *var*, *appl* and *abst* rules were already introduced in the simply typed λ-calculus. Here, they are just extended to also work for types and kinds. Keep in mind that s acts as a placeholder for both $*$ (types) and $\diamond$ (kinds). The *weak* rule has its name from the fact that if we expand a context, we weaken it. We see that we can add $x : C$ to the context, without changing $A : B$, if C is inhabitable, which we make sure by the axiom $C : s$, which really just means that C should not be a variable, but a type or kind. The *form* rule consists of four rules, as we can use all combinations of $*, \diamond \in s$.

$x : A : s_1$	$b : B : s_2$	(s_1, s_2)	$\lambda x : A, b$
$*$	$*$	$(*, *)$	term depending on a term
$\diamond$	$*$	$(\diamond, *)$	term depending on a type
$\diamond$	$\diamond$	$(\diamond, \diamond)$	type depending on a type
$*$	$\diamond$	$(*, \diamond)$	type depending on a term

Table 5: The four cases of the *form* rule.

This rule just states how one can build more complex types (arrow types) or kinds, considering the four cases, where we get different dependencies between terms and types. We want to remark that the column on the right shows the term after using the *abst* rule, which is why we write $\lambda x : A, b$. Last but not least, with the *conv* rule we make sure that if we have a term for an expression B, it is also a term for any expression B', such that $B =_\beta B'$. Note that most of these rules can already be introduced in lower systems, for example, the *form* rule is already part of $\lambda 2$, $\lambda \underline{\omega}$ and λP, just for different contexts.

While these rules seem quite abstract, they are actually enough to do constructive logic, and therefore fully implement natural deduction. We may state at this point that the two classical principles of the *excluded middle* ($P \vee \neg P = True$) and the *double negation* ($\neg(\neg P) \iff P$) cannot be derived from constructive calculus, but fortunately, they can be added as a definition, without disrupting the logic or leading to contradictions. With that, we have found a suitable system, on which we can build our foundations of mathematics. We will not go much more into technical details about the CoC, but we want to mention that CoC is Turing complete again, and that while *well-typedness* and *type checking* are still decidable in CoC, *term finding* is not anymore. By the Curry-Howard Isomorphism, if *term finding* was decidable in CoC, we could write a program that finds a proof for any proposition defined in predicate logic, which is not possible by Gödel's incompleteness theorem.

We are now finally able to talk about the logical foundation of LEAN. In LEAN, we use an extension of λC, namely the *calculus of inductive constructions* (*CiC* for short). For our scope, it is enough to understand that this system expands CoC by adding inductive types. More technical details can be found in [Pau15]. Another extra we get with LEAN is definitions. For example, we can simply type $P \vee Q$ without having to know how the $\vee$ operator is defined in λC. The λ-term for "or" would be:

$$\vee \equiv \lambda P : *, \lambda Q : *, \Pi R : *, (P \to R) \to (Q \to R) \to R$$

It can be shown that all connectives and quantifiers used for first-order logic can be implemented in LEAN using the calculus of constructions. We discuss the $\vee$ operator in detail, showing that it obeys the rules it should and then we list the remaining connectives and quantifiers defined in type-theoretic terms without more comments in Table 6. In the sequent calculus of predicate logic, "or" is defined using the following rules:

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\,(\vee\text{-intro-left})$$

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\,(\vee\text{-intro-right})$$

$$\frac{\Gamma \vdash P \vee Q \qquad P \vdash R \qquad Q \vdash R}{\Gamma, P \vee Q \vdash R}\,(\vee\text{-elim})$$

One can show that the λ-term which defines $\vee$, obeys all three rules. For instance, the $\vee$-elim rule can be obtained by applying *appl* twice. As a consequence, any proof involving $\vee$ can be derived using our definition.

Predicate logic	λ**C**
A is a set. P is a proposition.	$A : *$ $P : *$
$a \in A$ p proves P.	$a : A$ $p : P$
P is a predicate on A.	$P : A \to *$
$P \Rightarrow Q$	$P \to Q\ (= \Pi x : P, Q)$
$\bot$ $\neg P$	$\Pi \alpha : *, \alpha$ $P \to \bot$
$P \vee Q$ $P \wedge Q$	$\Pi R : *, (P \to R) \to (Q \to R) \to R$ $\Pi R : *, (P \to Q \to R) \to R$
$\forall x \in A, (P(x))$ $\exists x \in A, (P(x))$	$\Pi x : A, Px$ $\Pi R : *, ((\Pi x : A, (Px \to R)) \to R)$

Table 6: Predicate logic in λC.

The table above demonstrates how the logical connectives and quantifiers can be implemented as λ-terms.

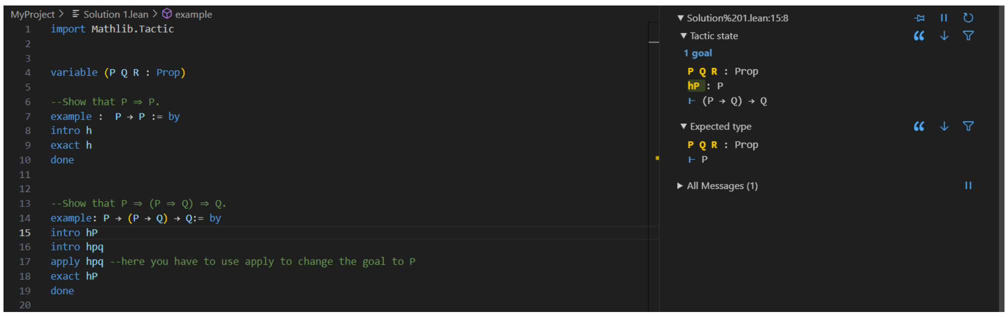
Finally, LEAN adds so-called tactics to the calculus of constructions, which allow us to do proofs in a more relatable way. We talk about these tactics soon.

Let us summarize what we did in this subchapter. We started by introducing type theory as an alternative to set theory, and then we learned about the λ-calculus. First, we only considered the untyped λ-calculus to understand the key idea behind it, namely to find a language to write mathematical functions and terms as programs. We soon realized that while this system is Turing complete, it still had some problems with unsettling terms and terms without a β-normal form. To solve this problem, we added types to the λ-calculus and found the system $\lambda{\to}$. With this new system, we solved old problems by introducing the notion of *legal terms*, but we ran into new problems, like the restriction of the identity function to a single type or the lack of more complex type structures. To tackle these new problems, we introduced the independent systems $\lambda 2$ and $\lambda\underline{\omega}$. We were reaching the end, but as we wanted to be able to do some predicate logic, where we would like to prove a predicate $P(x)$ depending on its input x, we still needed yet another system. As propositions P are considered as types and values x as terms by the PAT interpretation, we needed a system that introduces types depending on terms. We found that system to be λP. Our final step was to combine the three systems ($\lambda 2$, $\lambda\underline{\omega}$ and λP) to the calculus of constructions (λC). LEAN's logical foundation depends on an extended version of λC, where inductive types, definitions and tactics are added.

LEAN Interface

Now that we understand the foundation LEAN is based on, we would like to know how it operates. But before we can do that, we have to explain its interface.

There are several digital platforms, online too, one can use to program with LEAN. For this thesis, we downloaded and installed LEAN for Visual Studio Code (VS-Code). Instruction on how to do that can be found online [Com23a], or on our GitHub [Bot23]. Once we do this, we can start a LEAN project, and when we open it in VS-Code, it looks something like this:

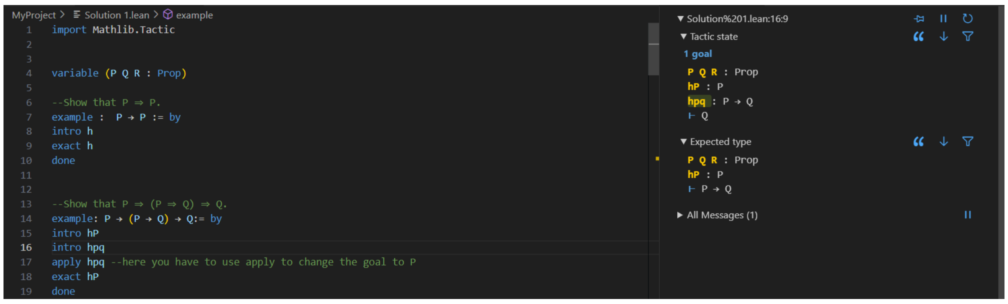


MyProject > Solution 1.lean > example

```
1   import Mathlib.Tactic
2
3
4   variable (P Q R : Prop)
5
6   --Show that P ⇒ P.
7   example :  P → P := by
8   intro h
9   exact h
10  done
11
12
13  --Show that P ⇒ (P ⇒ Q) ⇒ Q.
14  example: P → (P → Q) → Q:= by
15  intro hP
16  intro hpq
17  apply hpq --here you have to use apply to change the goal to P
18  exact hP
19  done
20
```

```
▼ Solution%201.lean:15:8
 ▼ Tactic state
  1 goal
  P Q R : Prop
  hP : P
  ⊢ (P → Q) → Q
 ▼ Expected type
  P Q R : Prop
  ⊢ P
▶ All Messages (1)
```

(a) LEAN interface at line 15.



MyProject > Solution 1.lean > example

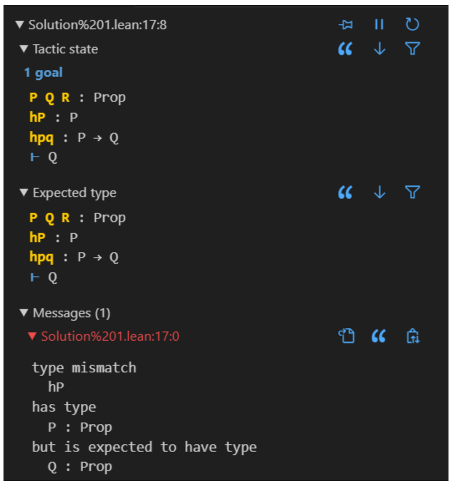
```
1   import Mathlib.Tactic
2
3
4   variable (P Q R : Prop)
5
6   --Show that P ⇒ P.
7   example :  P → P := by
8   intro h
9   exact h
10  done
11
12
13  --Show that P ⇒ (P ⇒ Q) ⇒ Q.
14  example: P → (P → Q) → Q:= by
15  intro hP
16  intro hpq
17  apply hpq --here you have to use apply to change the goal to P
18  exact hP
19  done
```

```
▼ Solution%201.lean:16:9
 ▼ Tactic state
  1 goal
  P Q R : Prop
  hP : P
  hpq : P → Q
  ⊢ Q
 ▼ Expected type
  P Q R : Prop
  hP : P
  ⊢ P → Q
▶ All Messages (1)
```

(b) LEAN interface at line 16.

Figure 3: The LEAN interface in VS-Code with the helpful infoview on the right side.

We do our proof writing on the left side of the interface. On top, in both figures, we can see that some packages were imported to use the tactics mode and other things. We talk about LEAN tactics later. Below, still in both figures of Figure 3 there is an example of a short proof that we wrote. Notice the way a proposition is written and how the following proof is constructed. On the right side, LEAN's magic happens. In the so-called *infoview*, we can see the state of the proof. In Figure 3, one sees that, depending on which line we are on the left, we see a different proof state in the infoview on the right. This is very helpful compared to doing proofs by hand, where one always needs to check what hypothesis we have and what still needs to be proved. If one makes a mistake or tries a tactic that is not allowed, an error message appears in the infoview, as can be seen Figure 4 below.

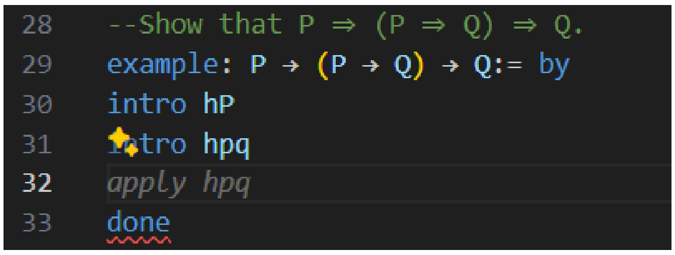


```
▼ Solution%201.lean:17:8
▼ Tactic state
1 goal
P Q R : Prop
hP : P
hpq : P → Q
⊢ Q

▼ Expected type
P Q R : Prop
hP : P
hpq : P → Q
⊢ Q

▼ Messages (1)
▼ Solution%201.lean:17:0

type mismatch
  hP
has type
  P : Prop
but is expected to have type
  Q : Prop
```

Figure 4: Error-message when trying to exact hP from the example above.

The error message not only says that there has been a mistake, but it can also say what kind of mistake happened. Depending on one's experience, these messages can help correct mistakes directly. Imagine if a piece of paper could do such a thing! The infoview can also show proposed theorems to solve a proof in certain situations. We talk more about this when we have a look at LEAN's tactics.

LEAN's automation already helps us finish a proof by giving feedback on the progress of the proofs, showing us the mistakes we made and even proposing theorems of its rich theorem collection in Mathlib. We can even take it to the next step when doing LEAN in VS-Code. With the rise of ChatGPT and other AIs, an AI able to make code suggestions, called GitHub Copilot, has been developed, which can be directly used in VS-Code. It is a VS-Code package that can be installed but needs a login. Fortunately, using a UZH account, one can use it for free.



```
--Show that P ⇒ (P ⇒ Q) ⇒ Q.
example: P → (P → Q) → Q:= by
intro hP
intro hpq
apply hpq
done
```

Figure 5: A suggestion for the next step generated by GitHub Copilot.

During the meetings with the students, the copilot was disabled, since it made some exercises too easy. However, it was still very useful in developing the solutions to the exercise sheets. It is important to mention that GitHub Copilot does not always give helpful lines of code and sometimes even proposes code that results in an error message, but it can help when one is stuck at a certain point in the proof, even if only by sparking an idea with a not working line of code.

Tactics and Theorems

When proving statements in LEAN, one can use so-called *tactics*, which are similar to instructions to proofs [Avi+23]. In this section, we list the most important tactics used in the levels and directly show an example of the tactic used in LEAN. In these examples, we often just show a small part of the whole proof. The complete proofs can be found on GitHub [Bot23]. Some of the tactics presented are directly related to natural deduction rules, and by the PAT interpretation, we can use the derivation rules we have learned from the calculus of constructions to build a term (the proof) corresponding to the given type (the proposition). For these tactics, we always present the natural deduction rule the tactic corresponds to, the proof using the tactics and the proof in term-style. Term-style proofs are the other way to prove statements in LEAN and they are closely related to the calculus of constructions. One can even use both methods combined to build a proof, but we do not consider that in this thesis. Apart from tactics and derivations rules, LEAN has also access to countless theorems and definitions which are implemented in a huge library called Mathlib. We have summarized the most important addition, subtraction, multiplication and division theorems in a "cheat sheet" that can be found on [Bot23]. But we won't list them all here since they would take up too much space.

Remark 2.2.27. We would actually need to write the natural deduction rules in sequent calculus since we introduced sequent calculus as the logical formalism before. However, natural deduction rules are easier to read, and for our purpose, it is enough to just give a proposition (type) P without giving it context and adding turntables.

intro tactic

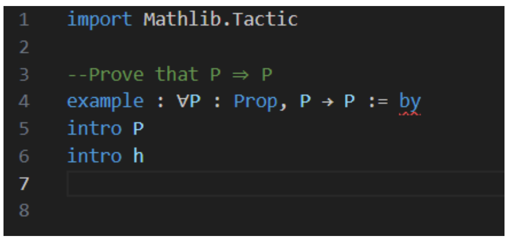
One of the very first things we want to do in a proof is to introduce the variables and hypotheses that we have. We can do that if we have an implication type in our goal. Written in natural deduction, introducing something looks like this.

$$\dfrac{\begin{array}{c}\text{Assume } P \\ \vdots \\ Q\end{array}}{P \to Q}\ (\Rightarrow \text{-intro})$$

What we are doing here is introducing P and from P we want to derive Q, seen above the line. This then gives the proof that P implies Q, seen below the line.

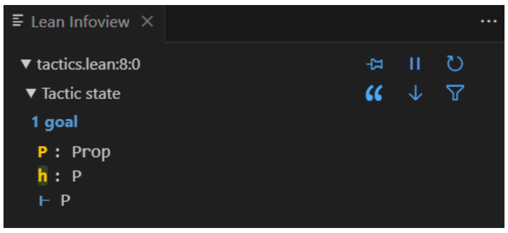
Definition 2.2.28. The **intro** or **intros** tactic can introduce variables or hypotheses.

Example 2.2.29.



```
import Mathlib.Tactic

--Prove that P ⇒ P
example : ∀P : Prop, P → P := by
intro P
intro h

```

(a) *intro* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:8:0
▼ Tactic state
1 goal
P : Prop
h : P
⊢ P
```

(b) LEAN infoview after using *intro*.

Figure 6: Using the *intro* tactic to prove that $P \Rightarrow P$ holds for any proposition.

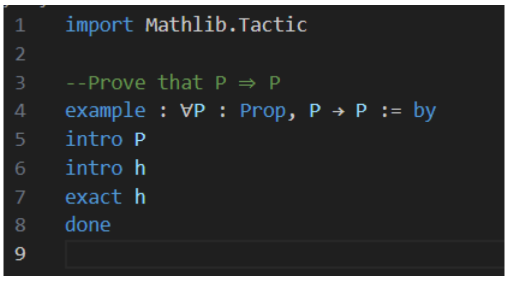
We see in the Figure 6 above, on the left, that the original goal is to prove that $\forall P :$ `Prop`, $P \to P$. Here, $P :$ `Prop` means that P is of type `Prop` (propositions). In the right picture, we see the remaining goal ($\vdash P$) and the two hypotheses we now have. They are $P :$ `Prop`, i.e. we now have an arbitrary proposition and $h : P$. The second means that we have a proof h for P independent of what proposition P is exactly.

exact tactic

To finish the proof in Figure 6, we need to somehow use the hypothesis $h : P$. We can do this with the following tactic.

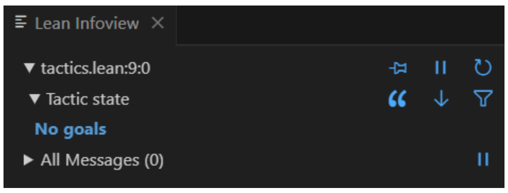
Definition 2.2.30. With the **exact** tactic we can apply a proposition or a proof to the given goal.

Example 2.2.31.



```
import Mathlib.Tactic

--Prove that P ⇒ P
example : ∀P : Prop, P → P := by
intro P
intro h
exact h
done

```

(a) *exact* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:9:0
 ▼ Tactic state
   No goals
▶ All Messages (0)
```

(b) LEAN infoview after using *exact*.

Figure 7: Using the *exact* tactic concludes the proof that $P \Rightarrow P$ holds for any proposition.

When we try to prove this without tactics, we do it as follows.

```
example : P → P := fun p : P => p
```

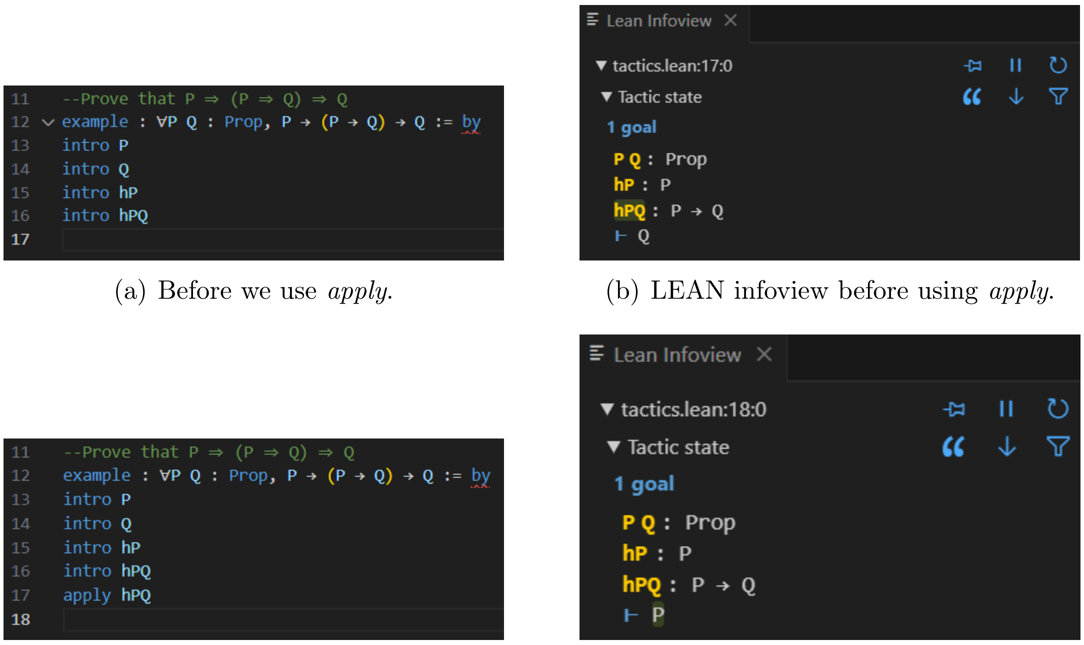
Here we introduce a function **fun** from P to P called p. When we are not in LEAN's tactic mode, this already finishes the proof.

apply tactic

Unfortunately it is not always that easy to prove a statement. We could for example have two implications in the statement we want to prove, e.g. $P \Rightarrow (P \Rightarrow Q) \Rightarrow Q$. Here we cannot simply use the *intro* tactic to introduce P and $P \Rightarrow Q$ and then exact something. Because to use the *exact* tactic, we need to have the same thing in the hypothesis and the goal. But we still have some tricks up our sleeves.

Definition 2.2.32. The **apply** tactic is used when a hypothesis with an implication is given and the goal is of the exact same form as the conclusion of the hypothesis.

Example 2.2.33.



```
--Prove that P ⇒ (P ⇒ Q) ⇒ Q
example : ∀P Q : Prop, P → (P → Q) → Q := by
intro P
intro Q
intro hP
intro hPQ
```

(a) Before we use *apply*.

```
Lean Infoview
tactics.lean:17:0
Tactic state
1 goal
P Q : Prop
hP : P
hPQ : P → Q
⊢ Q
```

(b) LEAN infoview before using *apply*.

```
--Prove that P ⇒ (P ⇒ Q) ⇒ Q
example : ∀P Q : Prop, P → (P → Q) → Q := by
intro P
intro Q
intro hP
intro hPQ
apply hPQ
```

(c) *apply* used in VS-Code.

```
Lean Infoview
tactics.lean:18:0
Tactic state
1 goal
P Q : Prop
hP : P
hPQ : P → Q
⊢ P
```

(d) LEAN infoview after using *apply*.

Figure 8: Using the *apply* tactic to deal with hypotheses including implications.

The right-hand side of the hypothesis hPQ is Q, which is the same as the goal. So we can use the *apply* tactic to change the goal to the left-hand side of the hypothesis, namely P. The *apply* tactic works with the following principle: If we have an implication, e.g. $P \Rightarrow Q$ as a hypothesis, i.e. we know that the implication holds, where hPQ is the proof for $P \Rightarrow Q$, then it is enough to show the left-hand side of the implication to prove the right-hand side. So we might as well change the goal from the right-hand side to the left-hand side. In natural deduction, this rule looks like this.

$$\frac{P \Rightarrow Q \qquad P}{Q} \ (\Rightarrow\text{-elim})$$

Remark 2.2.34. One can forget about the *apply* tactic and use *exact hPQ hP* instead.

Proving this without tactics would look like this.

```
example : P → (P → Q) → Q :=
  fun p : P =>
  fun f : P → Q =>
  f p
```

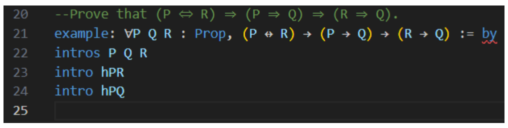
One may have already noticed that the structure of the proof changes when we do it without tactics. Tactics can only be used inside a by-done environment. We see that in this case, we did not use *apply* explicitly.

rw tactic

We are a bit more in luck if we have if-and-only-if statements as hypotheses. Then we can decide which side we would like to use.

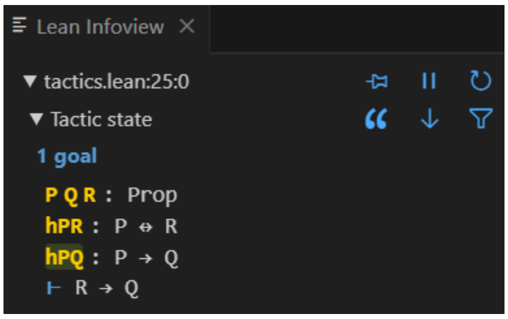
Definition 2.2.35. When we have an if-and-only-if statement as a hypothesis or in Mathlib, we can use the **rw** (rewrite) tactic to change the statement accordingly. We can even use **rw at** to change something in a given hypothesis. If we want to rw something several times, we can use **repeat rw**.

Example 2.2.36.



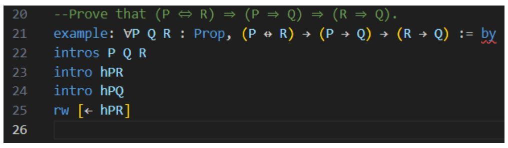
```
--Prove that (P ⇔ R) ⇒ (P ⇒ Q) ⇒ (R ⇒ Q).
example: ∀P Q R : Prop, (P ↔ R) → (P → Q) → (R → Q) := by
intros P Q R
intro hPR
intro hPQ

```

(a) Before we use *rw*.



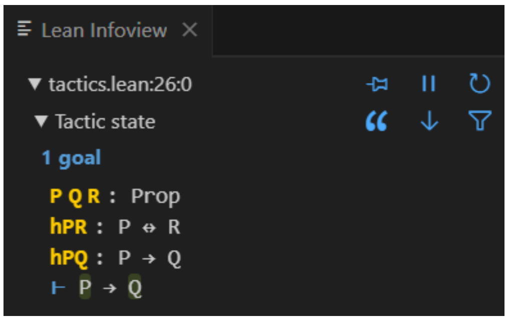
```
Lean Infoview
▼ tactics.lean:25:0
▼ Tactic state
1 goal
P Q R : Prop
hPR : P ↔ R
hPQ : P → Q
⊢ R → Q
```

(b) LEAN infoview before using *rw*.



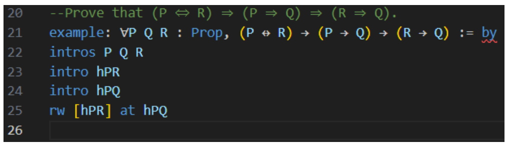
```
--Prove that (P ⇔ R) ⇒ (P ⇒ Q) ⇒ (R ⇒ Q).
example: ∀P Q R : Prop, (P ↔ R) → (P → Q) → (R → Q) := by
intros P Q R
intro hPR
intro hPQ
rw [← hPR]

```

(c) *rw* used in the goal.



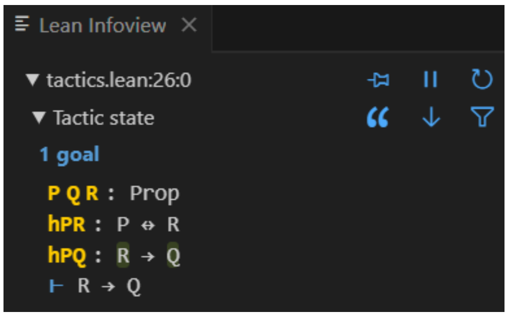
```
Lean Infoview
▼ tactics.lean:26:0
▼ Tactic state
1 goal
P Q R : Prop
hPR : P ↔ R
hPQ : P → Q
⊢ P → Q
```

(d) LEAN infoview after using *rw* in the goal.



```
--Prove that (P ⇔ R) ⇒ (P ⇒ Q) ⇒ (R ⇒ Q).
example: ∀P Q R : Prop, (P ↔ R) → (P → Q) → (R → Q) := by
intros P Q R
intro hPR
intro hPQ
rw [hPR] at hPQ
```

(e) *rw* used in the hypothesis.



```
Lean Infoview
▼ tactics.lean:26:0
▼ Tactic state
1 goal
P Q R : Prop
hPR : P ↔ R
hPQ : R → Q
⊢ R → Q
```

(f) LEAN infoview after using *rw* in the hypothesis.

Figure 9: Using the *rw* tactic to change between P and R.

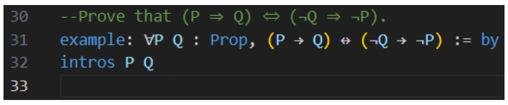
In Figure 9 we show the difference between using the *rw* tactic in the goal or the hypothesis. Note that for the first case, we need to add a left arrow ($\leftarrow$), since we want to change R to P and not the other way around. In this explicit example, we would not recommend changing the hypothesis, since the name hPQ will not make sense after changing P to R. We could also just name our hypothesis differently, such that it does not depend on P or R.

constructor tactic

There are often if-and-only-if statements that we would like to prove. On paper, we simply prove one implication and then the other. In LEAN, one can do the same thing.

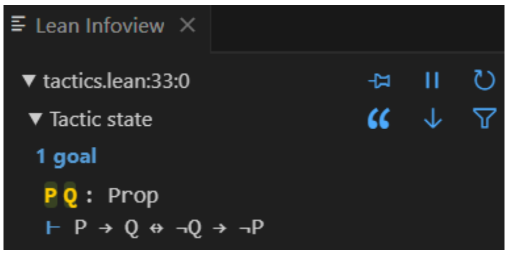
Definition 2.2.37. The **constructor** tactic is used to split an if-and-only-if goal into two subgoals where we need to prove both implications.

Example 2.2.38.



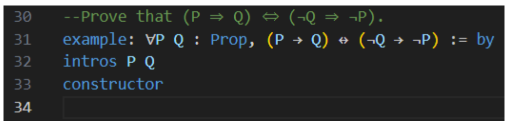
```
--Prove that (P ⇒ Q) ⇔ (¬Q ⇒ ¬P).
example: ∀P Q : Prop, (P → Q) ↔ (¬Q → ¬P) := by
intros P Q
```

(a) Before we use *constructor*.



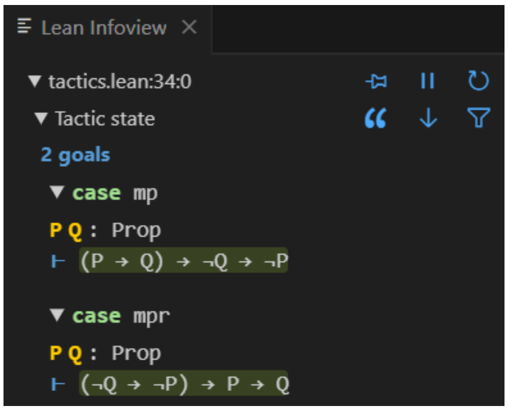
```
Lean Infoview
▼ tactics.lean:33:0
▼ Tactic state
1 goal
P Q : Prop
⊢ P → Q ↔ ¬Q → ¬P
```

(b) LEAN infoview before using *constructor*.



```
--Prove that (P ⇒ Q) ⇔ (¬Q ⇒ ¬P).
example: ∀P Q : Prop, (P → Q) ↔ (¬Q → ¬P) := by
intros P Q
constructor
```

(c) *constructor* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:34:0
▼ Tactic state
2 goals
▼ case mp
P Q : Prop
⊢ (P → Q) → ¬Q → ¬P

▼ case mpr
P Q : Prop
⊢ (¬Q → ¬P) → P → Q
```

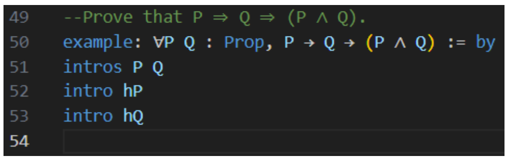
(d) LEAN infoview after using *constructor*.

Figure 10: The *constructor* tactic splits a goal into two subgoals for both implications.

The *constructor* tactic can also be used when we have a ∧ in the goal. Then this tactic corresponds to the ∧-introduction rule:

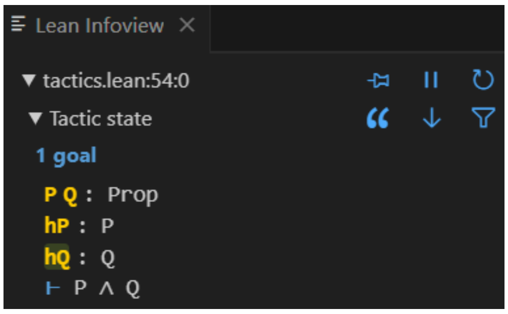
$$\frac{P \qquad Q}{P \wedge Q}\ (\wedge\text{-intro})$$

Example 2.2.39.



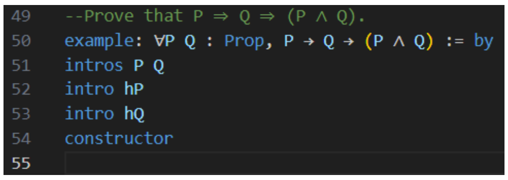
```
--Prove that P ⇒ Q ⇒ (P ∧ Q).
example: ∀P Q : Prop, P → Q → (P ∧ Q) := by
intros P Q
intro hP
intro hQ
```

(a) Before we use *constructor*.



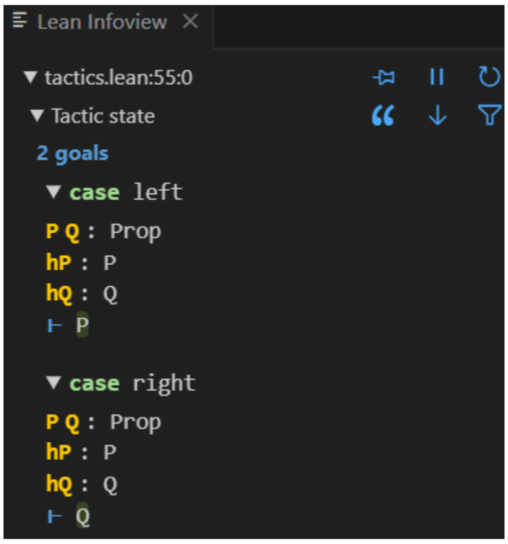
```
Lean Infoview
▼ tactics.lean:54:0
▼ Tactic state
1 goal
P Q : Prop
hP : P
hQ : Q
⊢ P ∧ Q
```

(b) LEAN infoview before using *constructor*.



```
--Prove that P ⇒ Q ⇒ (P ∧ Q).
example: ∀P Q : Prop, P → Q → (P ∧ Q) := by
intros P Q
intro hP
intro hQ
constructor

```

(c) *constructor* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:55:0
▼ Tactic state
2 goals
▼ case left
P Q : Prop
hP : P
hQ : Q
⊢ P

▼ case right
P Q : Prop
hP : P
hQ : Q
⊢ Q
```

(d) LEAN infoview after using *constructor*.

Figure 11: The *constructor* tactic splits a goal into two subgoals for both sides of the $\wedge$ operator.

After we split the proof into two subgoals, we can simply use exact hP and exact hQ to finish the proof.

When we are not in the tactics mode, we use the same expression as in the sequent calculus:

```
example : P → Q → (P ∧ Q) :=
  fun p =>
  fun q =>
  And.intro p q
```

With the *constructor* tactic, we can already see a huge advantage that tactics have. While the $\wedge$-introduction rule only works for conjunction terms, the *constructor* tactic can also split if-and-only-if statements. LEAN knows by itself which case is treated.

left and right tactics

Let us see what changes if we have a disjunction in the goal.

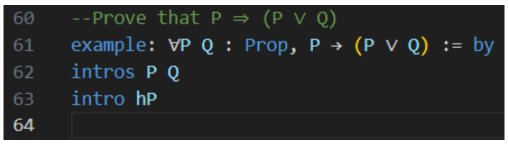
Definition 2.2.40. Contrary to proofs with $\wedge$ in the goal, we only need to prove one of the two propositions when combined with an $\vee$. We can decide ourselves which side we want to prove using the **left** or **right** tactic.

In sequent calculus, the corresponding deduction rule is called the $\vee$-introduction-left/right rule:

$$\frac{P}{P \vee Q} \; (\vee\text{-intro-left})$$

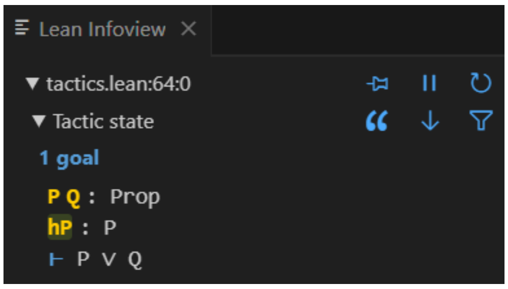
$$\frac{Q}{P \vee Q} \; (\vee\text{-intro-right})$$

Example 2.2.41.



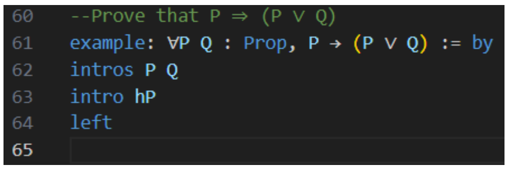
```
--Prove that P ⇒ (P ∨ Q)
example: ∀P Q : Prop, P → (P ∨ Q) := by
intros P Q
intro hP
```

(a) Before we use the *left* tactic.



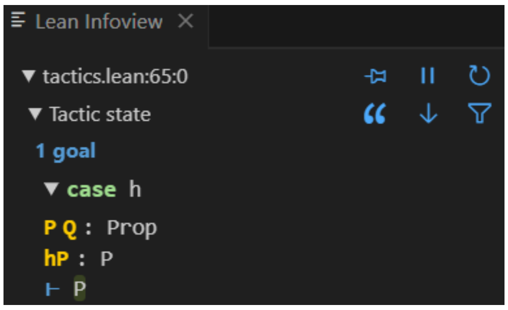
```
Lean Infoview
▼ tactics.lean:64:0
▼ Tactic state
1 goal
P Q : Prop
hP : P
⊢ P ∨ Q
```

(b) LEAN infoview before using *left*.



```
--Prove that P ⇒ (P ∨ Q)
example: ∀P Q : Prop, P → (P ∨ Q) := by
intros P Q
intro hP
left
```

(c) *left* tactic used in VS-Code.



```
Lean Infoview
▼ tactics.lean:65:0
▼ Tactic state
1 goal
▼ case h
P Q : Prop
hP : P
⊢ P
```

(d) LEAN infoview after using the *left* tactic.

Figure 12: The *left* tactic lets us prove only the left side of a statement.

Since we know that P has a proof, we choose *left*. We could use *right* if we would know a proof for Q. We show this case for the term-style proof in LEAN.

```
example : Q → (P ∨ Q) :=
  fun q : Q =>
  Or.inr q
```

Again we see that if we do not use tactics, the names of the rules we use are similar to the ones from sequent calculus.

cases' tactic

Disjunction (∨) and conjunction (∧) terms can also appear as a hypotheses. Think about how the two differ when the statements are split.

Definition 2.2.42. Whenever there is an ∧ or an ∨ in one of the hypotheses instead of in the goal, we can use the **cases'** tactic. For a logical "and", this gives the same proof term with two hypotheses. Whereas for the logical "or", one has to prove the same statement twice, once with the left-hand side of the hypothesis and once with the right-hand side.

This tactic corresponds to two deduction rules at once, the ∧-elimination rules and the ∨-elimination rule:

$$\frac{P \wedge Q}{P}\ (\wedge\text{-elim-left})$$

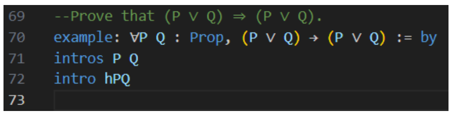
$$\frac{P \wedge Q}{Q}\ (\wedge\text{-elim-right})$$

$$\frac{P \vee Q \qquad P \Rightarrow R \qquad Q \Rightarrow R}{R}\ (\vee\text{-elim})$$

We see that with the ∧-elimination rule we can derive P and we can derive Q. When using the *cases'* tactic on a conjunction hypothesis, LEAN does both and we get two hypotheses. The elimination rule for disjunction is a bit more complicated. Assume that we have some statement R and a proof for $P \vee Q$. In order to show that $P \vee Q$ implies R, we need to prove that P implies R and that Q implies R. We need to have them both imply R, as we want to be able to decide if we want to use P or Q to imply R. This is why when we have $P \vee Q$ in the hypothesis, we have to prove two statements. Namely $P \Rightarrow R$ and $Q \Rightarrow R$.

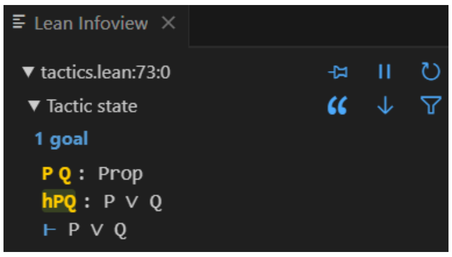
LEAN can decide from the context, which of the two deduction rules needs to be applied.

Example 2.2.43.



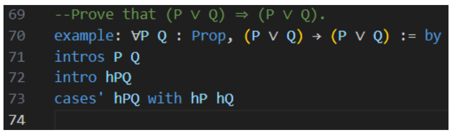
```
--Prove that (P ∨ Q) ⇒ (P ∨ Q).
example: ∀P Q : Prop, (P ∨ Q) → (P ∨ Q) := by
intros P Q
intro hPQ
```

(a) Before we use *cases'*.



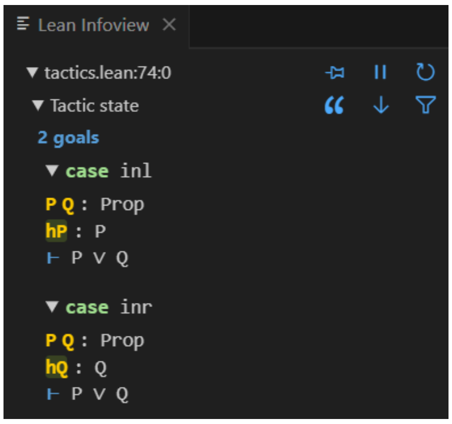
```
Lean Infoview
▼ tactics.lean:73:0
 ▼ Tactic state
  1 goal
  P Q : Prop
  hPQ : P ∨ Q
  ⊢ P ∨ Q
```

(b) LEAN infoview before using *cases'*.



```
--Prove that (P ∨ Q) ⇒ (P ∨ Q).
example: ∀P Q : Prop, (P ∨ Q) → (P ∨ Q) := by
intros P Q
intro hPQ
cases' hPQ with hP hQ
```

(c) *cases'* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:74:0
 ▼ Tactic state
  2 goals
   ▼ case inl
   P Q : Prop
   hP : P
   ⊢ P ∨ Q

   ▼ case inr
   P Q : Prop
   hQ : Q
   ⊢ P ∨ Q
```

(d) LEAN infoview after using *cases'*.

Figure 13: *cases'* applied on a hypothesis with “or” gives us two subgoals to prove.

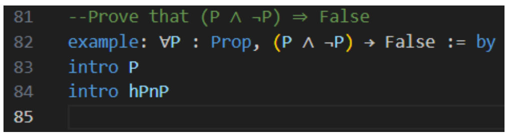
Obviously, we could just use *exact hPQ*, but this example is to illustrate the effect that *cases'* has on a logical or-statement. Note that we can give the two newly created hypotheses individual names.

Without the tactics mode, we must differentiate between a conjunction and a disjunction in the proof. For disjunction, we prove it as follows.

```
example : (P ∨ Q) → (Q ∨ P) :=
  fun h : P ∨ Q =>
  Or.elim h
    (fun p : P => Or.inr p)
    (fun q : Q => Or.inl q)
```

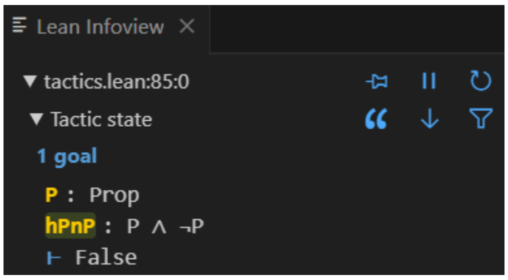
A big advantage of the tactic mode is that it shows what our current goal is. This can help a lot in cases where we have to prove two things independently, like in the example above.

Example 2.2.44.



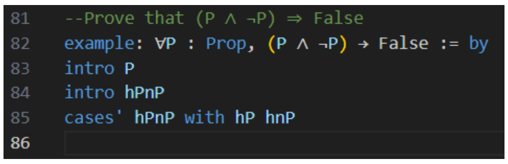
```
--Prove that (P ∧ ¬P) ⇒ False
example: ∀P : Prop, (P ∧ ¬P) → False := by
intro P
intro hPnP
```

(a) Before we use *cases'*.



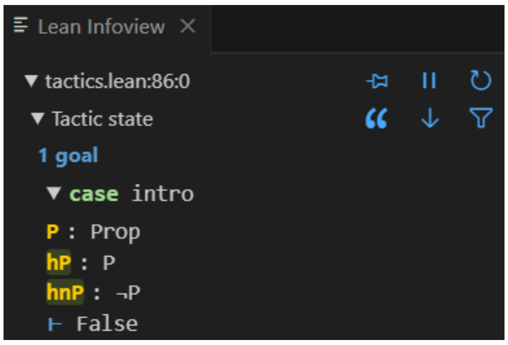
```
Lean Infoview
tactics.lean:85:0
Tactic state
1 goal
P : Prop
hPnP : P ∧ ¬P
⊢ False
```

(b) LEAN infoview before using *cases'*.



```
--Prove that (P ∧ ¬P) ⇒ False
example: ∀P : Prop, (P ∧ ¬P) → False := by
intro P
intro hPnP
cases' hPnP with hP hnP
```

(c) *cases'* used in VS-Code.



```
Lean Infoview
tactics.lean:86:0
Tactic state
1 goal
case intro
P : Prop
hP : P
hnP : ¬P
⊢ False
```

(d) LEAN infoview after using *cases'*.

Figure 14: *cases'* applied on a hypothesis with "and" gives us two hypotheses to prove one goal.

Note how here we do not change the goal when using the cases' tactic. Keep in mind that $\neg P$ and $P \Rightarrow False$ are logically equal, as one can check this with a truth table. In LEAN, $\neg P$ is defined as $P \Rightarrow False$ and can be used accordingly. To finish the proof, we *apply* hnP, which turns the goal into P, and *exact* hP.

We can do the same without tactics. Notice that here we do not use *apply* again:

```
example : (P ∧ ¬P) → False :=
  fun h : P ∧ ¬P =>
  And.right h (And.left h)
```

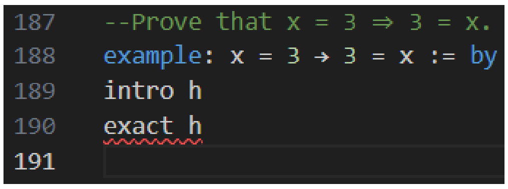
While it is convenient that the rules are called *And.right* and *And.left*, it should be mentioned that we never see the two hypotheses hP and hnP in this case, contrary to when we use tactics.

symm tactic

LEAN is very precise when checking proofs. For example, if the hypothesis is $h : x = 3$, we cannot prove the goal $\vdash 3 = x$ using exact h. One needs to switch the hypothesis or the goal.

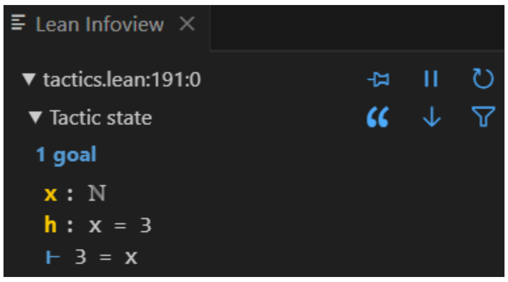
Definition 2.2.45. The **symm** tactic lets us interchange the left- and right-hand side of an equality in any goal or hypothesis.

Example 2.2.46.



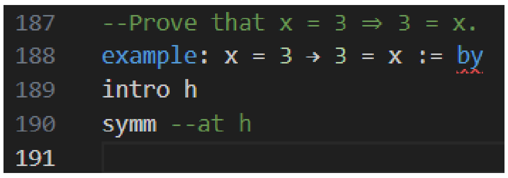
```
--Prove that x = 3 ⇒ 3 = x.
example: x = 3 → 3 = x := by
intro h
exact h
```

(a) Before we use *symm*.



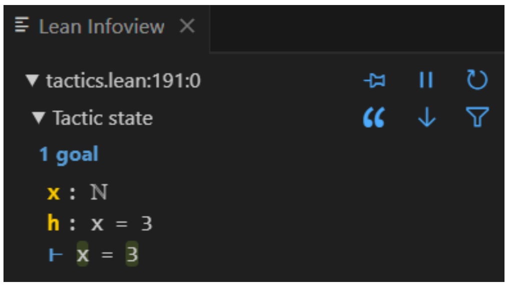
```
Lean Infoview
▼ tactics.lean:191:0
▼ Tactic state
1 goal
x : N
h : x = 3
⊢ 3 = x
```

(b) LEAN infoview before *symm*.



```
--Prove that x = 3 ⇒ 3 = x.
example: x = 3 → 3 = x := by
intro h
symm --at h
```

(c) *symm* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:191:0
▼ Tactic state
1 goal
x : N
h : x = 3
⊢ x = 3
```

(d) LEAN infoview after using *symm*.

Figure 15: Without *symm*, the proof cannot be concluded.

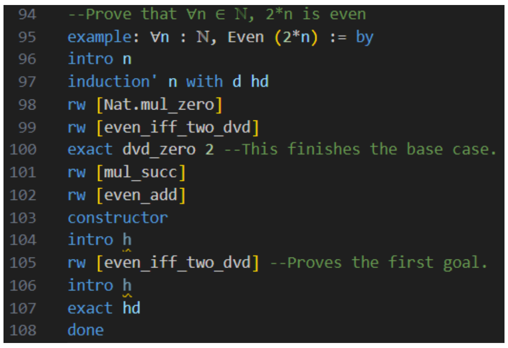
We could also change the equality in the hypothesis h, then the new hypothesis would be $h : 3 = x$. Exactly like the goal.

induction' tactic

We talk about natural induction in Chapter 2.1. Natural induction is something that LEAN knows how to do.

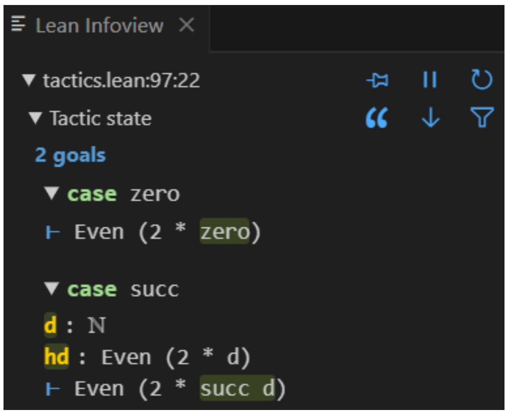
Definition 2.2.47. Using the **induction'** tactic, we can start doing a proof by induction over a variable. There is even the possibility of making a proof by strong induction.

Example 2.2.48.



```
--Prove that ∀n ∈ ℕ, 2*n is even
example: ∀n : ℕ, Even (2*n) := by
intro n
induction' n with d hd
rw [Nat.mul_zero]
rw [even_iff_two_dvd]
exact dvd_zero 2 --This finishes the base case.
rw [mul_succ]
rw [even_add]
constructor
intro h
rw [even_iff_two_dvd] --Proves the first goal.
intro h
exact hd
done
```

(a) A proof using natural induction.



```
▼ tactics.lean:97:22
 ▼ Tactic state
  2 goals
   ▼ case zero
   ⊢ Even (2 * zero)

   ▼ case succ
   d : ℕ
   hd : Even (2 * d)
   ⊢ Even (2 * succ d)
```

(b) LEAN infoview after using *induction'*.

Figure 16: Natural induction in LEAN.

Even though this is a trivial statement, we use quite specific theorems to prove it. We elaborate on that in Chapter 5.2.

The proof by the strong induction method was used to solve a question from one of the exercise sheets. Interested readers can have a look at that proof in Level 3, Exercise 3.2 in [Bot23].

have and let tactics

Definition 2.2.49. With **have** and **let** we can construct our own functions or hypotheses. We then have to prove those first before we can continue with our proof.

In the scope of the derivation rules seen in the calculus of constructions before, *have* and *let* would correspond to adding statements to the context Γ.

Example 2.2.50.

```
/-Use the method of direct proof to prove the following statements.
Let x, y ∈ ℝ. If x^2 + 5y = y^2 + 5x, then x = y or x + y = 5.-/
example (x y : ℝ) :  (x ^ 2 + 5 * y = y ^ 2 + 5 * x) → ((x = y) ∨ (x +
    y = 5)) := by
intro h
have h3 : (x - y) * (x + y - 5) = 0
ring_nf
rw [← sub_eq_zero] at h
ring_nf at h
exact h
rw [mul_eq_zero] at h3
cases' h3 with h1 h2
left
rw [sub_eq_zero] at h1
exact h1
right
rw [sub_eq_zero] at h2
exact h2
done
```

If we would not use the *have* tactic here, we would need to manually transform $x^2 + 5y = y^2 + 5x$ to $(x - y) \cdot (x + y - 5) = 0$, which needs a lot of *rw* tactics.

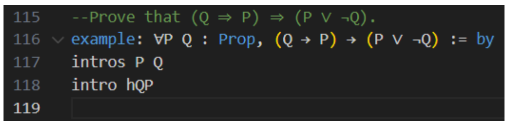
by_cases tactic

Sometimes we need to make a case distinction to solve a proof. LEAN has a tactic for that.

Definition 2.2.51. If we want to make a case distinction in one of our proofs, we can use the **by_cases** tactic to get two goals instead of one. Once with the hypothesis h and once with the hypothesis $\neg h$.

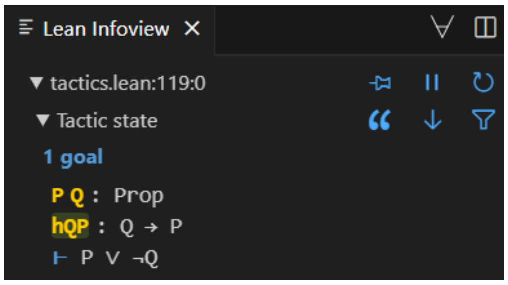
The *by_cases* tactic corresponds to the law of the excluded middle, which is not constructive! It states that $P \vee \neg P$ is a true statement. We cannot derive this from the CoC deduction rules, but we can add it as an axiom and use it as a tactic in LEAN.

Example 2.2.52.



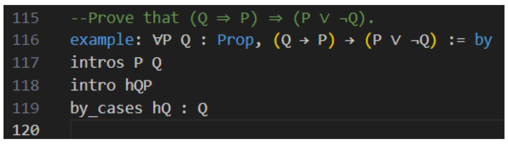
```
--Prove that (Q ⇒ P) ⇒ (P ∨ ¬Q).
example: ∀P Q : Prop, (Q → P) → (P ∨ ¬Q) := by
intros P Q
intro hQP
```

(a) Before we use *by_ cases*.



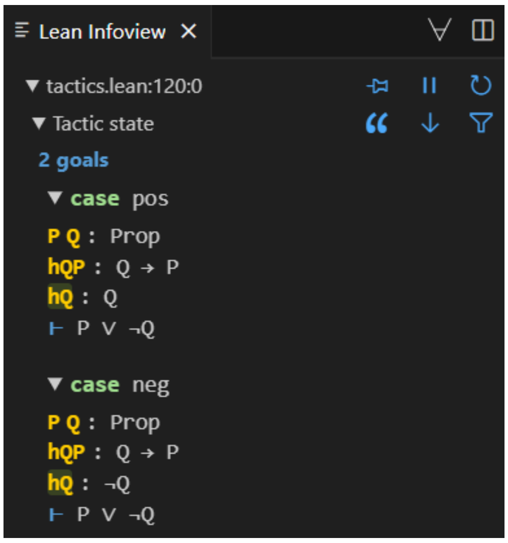
```
▼ tactics.lean:119:0
▼ Tactic state
1 goal
P Q : Prop
hQP : Q → P
⊢ P ∨ ¬Q
```

(b) LEAN infoview before *by_ cases*.



```
--Prove that (Q ⇒ P) ⇒ (P ∨ ¬Q).
example: ∀P Q : Prop, (Q → P) → (P ∨ ¬Q) := by
intros P Q
intro hQP
by_cases hQ : Q
```

(c) *by_ cases* used in VS-Code.



```
▼ tactics.lean:120:0
▼ Tactic state
2 goals
▼ case pos
P Q : Prop
hQP : Q → P
hQ : Q
⊢ P ∨ ¬Q

▼ case neg
P Q : Prop
hQP : Q → P
hQ : ¬Q
⊢ P ∨ ¬Q
```

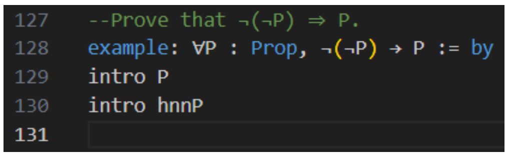
(d) LEAN infoview after using *by_ cases*.

Figure 17: We need to make a case distinction to prove this.

by_contra tactic

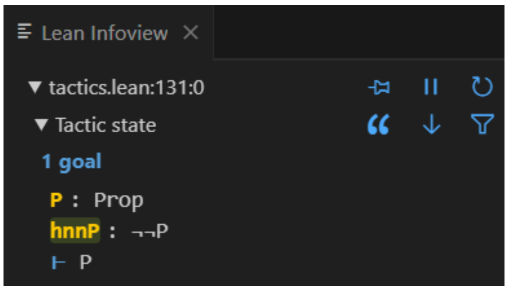
Definition 2.2.53. We can change our goal to *False* and add the goal as a negated hypothesis using the **by_contra** tactic. This is particularly useful for proofs by contradiction.

Example 2.2.54.



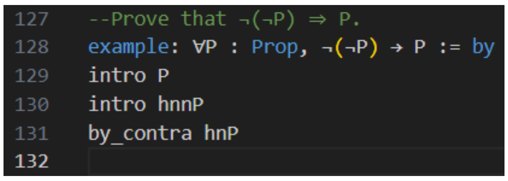
```
--Prove that ¬(¬P) ⇒ P.
example: ∀P : Prop, ¬(¬P) → P := by
intro P
intro hnnP
```

(a) Before we use *by_contra*.



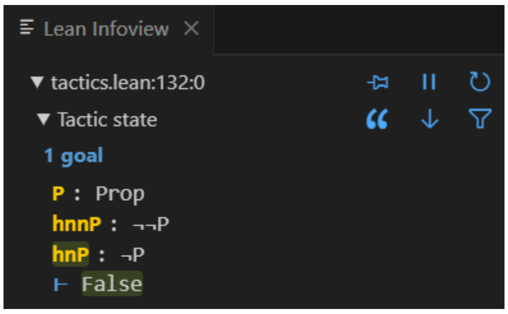
```
Lean Infoview
▼ tactics.lean:131:0
▼ Tactic state
1 goal
P : Prop
hnnP : ¬¬P
⊢ P
```

(b) LEAN infoview before *by_contra*.



```
--Prove that ¬(¬P) ⇒ P.
example: ∀P : Prop, ¬(¬P) → P := by
intro P
intro hnnP
by_contra hnP
```

(c) *by_contra* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:132:0
▼ Tactic state
1 goal
P : Prop
hnnP : ¬¬P
hnP : ¬P
⊢ False
```

(d) LEAN infoview after using *by_contra*.

Figure 18: Assume that the implication does not hold and find a contradiction proof.

What we have proven right here is the rule of double negation, which in fact goes in both ways, which is not a rule in constructive logic.

In the following, we have three tactics that can prove numerical expression in the blink of an eye.

norm_num tactic

Definition 2.2.55. Whenever there is a numerical expression that needs to be proved, we can use the **norm_num** tactic. This tactic can solve all kinds of statements without variables.

Example 2.2.56.

```
--Prove that 2 + 2 = 4.
example: 2 + 2 = 4 := by
norm_num
done
```

We can copy-paste this code into VS-Code to see that it compiles correctly.

ring_nf tactic

Definition 2.2.57. As long as our goal contains expressions with only ring arithmetic, we can use the **ring_nf** tactic to prove the statement.

Example 2.2.58.

```
--Prove that (x+y)^2 = x^2 + 2xy + y^2.
example: ∀x y : ℝ, (x+y)^2 = x^2 + 2*x*y + y^2 := by
intros x y
ring_nf
done
```

linarith and nlinarith tactics

Definition 2.2.59. Inequalities which are trivial considering our hypotheses can be proved with the tactic **linarith**. If the inequality contains non-linear terms, we can use **nlinarith** instead.

Example 2.2.60.

```
--Prove that x ≥ 3 ⇒ x ≥ 2.
example: ∀x : ℝ, x ≥ 3 → x ≥ 2 := by
intro x
intro h
linarith
done
```

Example 2.2.61. This implication here cannot be proved by *linarith*. Instead, we use *nlinarith*.

```
--Prove that x ≥ 3 ⇒ x^2 ≥ 9.
example: ∀x : ℝ, x ≥ 3 → x^2 ≥ 9 := by
intro x
intro h
nlinarith
done
```

simp tactic

Sometimes we are in a situation where all looks lost. Our hypotheses and/or our goals are a mess, but we are certain that we are very close. In that case, LEAN's automated theorem proving comes into play.

Definition 2.2.62. The most convenient tactic is the **simp** tactic. This tactic checks all theorems in the Mathlib library with the @simp attribute and tries to apply them. We can also use **simp at** to change a hypothesis or **simp*** to check everything (hypotheses and goals). If we would like to know what the **simp** tactic did, we just type **simp?**.

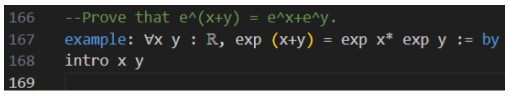
The possibility of using *simp?* to see what happened makes one feel less like a cheater. The *simp* tactic is certainly something one should keep in mind.

exact? and apply? tactics

If we would like to be less careless with simplifying statements but still do not know how to go on, we can use one of the following tactics.

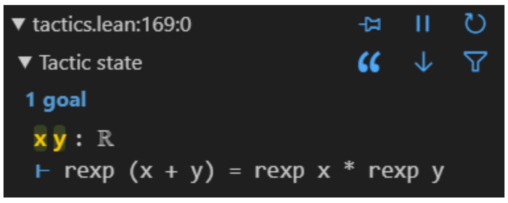
Definition 2.2.63. Two other useful tactics to help us continue to finish a proof are the **apply?** and **exact?** tactics. The first tries to find a theorem in the library that has the goal of a conclusion. The latter searches for a theorem that can directly prove the statement.

Example 2.2.64.



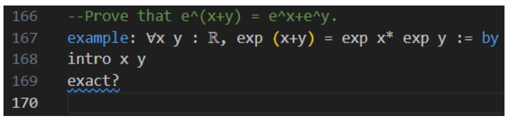
```
--Prove that e^(x+y) = e^x+e^y.
example: ∀x y : ℝ, exp (x+y) = exp x* exp y := by
intro x y

```

(a) Before we use *exact?*.



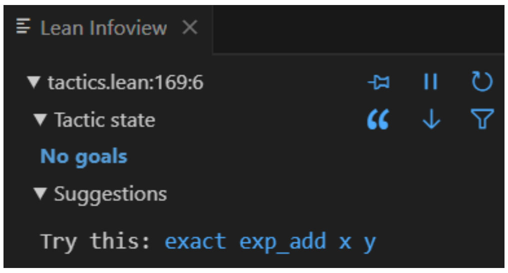
```
▼ tactics.lean:169:0
 ▼ Tactic state
  1 goal
   x y : ℝ
   ⊢ rexp (x + y) = rexp x * rexp y
```

(b) LEAN infoview before using *exact?*.



```
--Prove that e^(x+y) = e^x+e^y.
example: ∀x y : ℝ, exp (x+y) = exp x* exp y := by
intro x y
exact?
```

(c) *exact?* used in VS-Code.



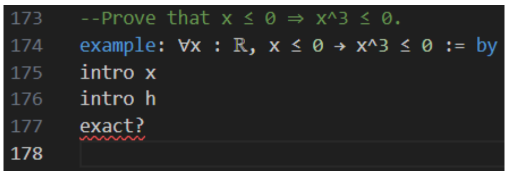
```
Lean Infoview
▼ tactics.lean:169:6
▼ Tactic state
No goals
▼ Suggestions
Try this: exact exp_add x y
```

(d) LEAN infoview after using *exact?*.

Figure 19: *exact?* used to find a theorem in Mathlib proving the desired statement.

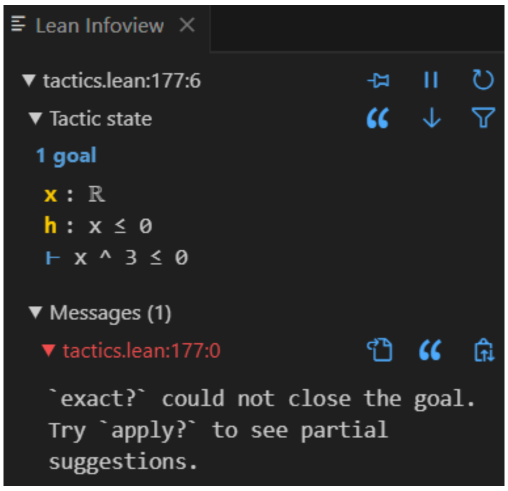
Unfortunately there is not a theorem for each statement we want to prove, especially, if we need to prove several steps. However, LEAN can also help us find these steps.

Example 2.2.65.



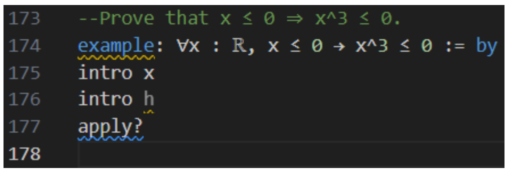
```
--Prove that x ≤ 0 ⇒ x^3 ≤ 0.
example: ∀x : ℝ, x ≤ 0 → x^3 ≤ 0 := by
intro x
intro h
exact?
```

(a) Trying to use *exact?*.



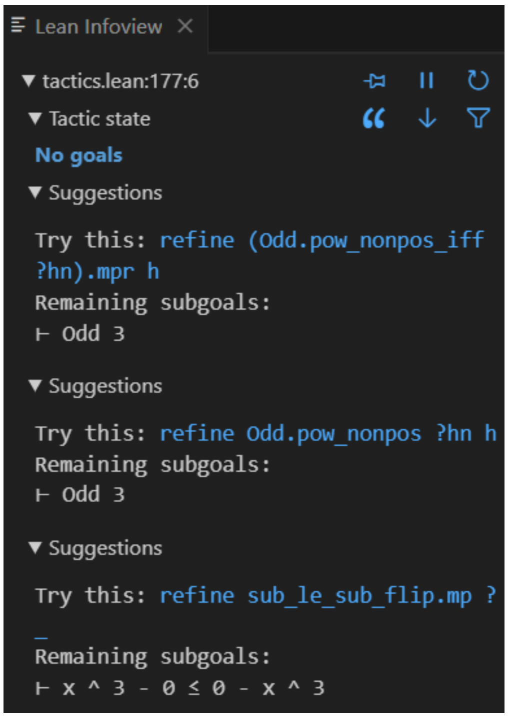
```
Lean Infoview
▼ tactics.lean:177:6
▼ Tactic state
1 goal
x : ℝ
h : x ≤ 0
⊢ x ^ 3 ≤ 0

▼ Messages (1)
▼ tactics.lean:177:0
`exact?` could not close the goal.
Try `apply?` to see partial
suggestions.
```

(b) LEAN infoview when using *exact?*.



```
--Prove that x ≤ 0 ⇒ x^3 ≤ 0.
example: ∀x : ℝ, x ≤ 0 → x^3 ≤ 0 := by
  intro x
  intro h
  apply?
```

(c) *apply?* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:177:6
  ▼ Tactic state
    No goals
  ▼ Suggestions
    Try this: refine (Odd.pow_nonpos_iff
    ?hn).mpr h
    Remaining subgoals:
    ⊢ Odd 3
  ▼ Suggestions
    Try this: refine Odd.pow_nonpos ?hn h
    Remaining subgoals:
    ⊢ Odd 3
  ▼ Suggestions
    Try this: refine sub_le_sub_flip.mp ?
    _
    Remaining subgoals:
    ⊢ x ^ 3 - 0 ≤ 0 - x ^ 3
```

(d) LEAN infoview after using *apply?*.

Figure 20: *apply?* used to find a theorem in Mathlib that changes the goal.

LEAN gives many more suggestions than the ones shown in Figure 20. With one of the suggestions shown, we can finish the proof.

use tactic

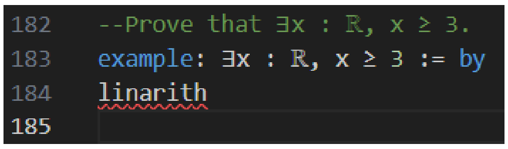
We have not talked a lot about quantifiers until now. What do we do when we have an ∃ in our goal or a ∀ symbol in our hypothesis?

Definition 2.2.66. Whenever there is an ∃ quantifier, we can use the **use** tactic to introduce the example we want to prove the statement with.

This tactic corresponds to the ∃-introduction rule:

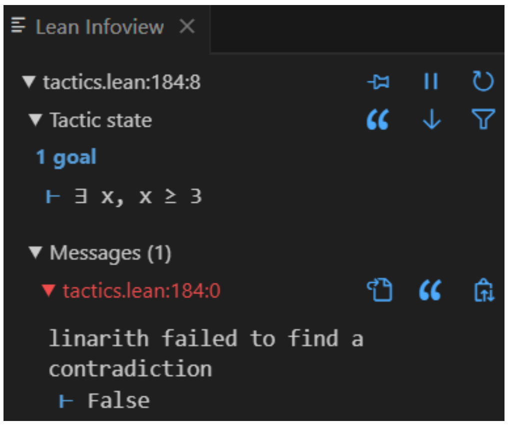
$$\frac{a \in A \qquad P(a)}{\exists x \in A, P(x)}\ (\exists\text{-intro})$$

Example 2.2.67.



```
--Prove that ∃x : ℝ, x ≥ 3.
example: ∃x : ℝ, x ≥ 3 := by
linarith
```

(a) Before we use *use.*



Lean Infoview

▼ tactics.lean:184:8

▼ Tactic state

1 goal

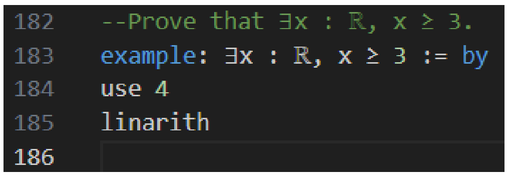
```
⊢ ∃ x, x ≥ 3
```

▼ Messages (1)

▼ tactics.lean:184:0

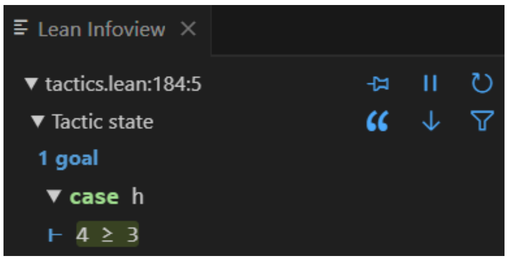
```
linarith failed to find a
contradiction
 ⊢ False
```

(b) LEAN infoview before using *use.*



```
--Prove that ∃x : ℝ, x ≥ 3.
example: ∃x : ℝ, x ≥ 3 := by
use 4
linarith
```

(c) *use* used in VS-Code.



Lean Infoview

▼ tactics.lean:184:5

▼ Tactic state

1 goal

▼ case h

```
⊢ 4 ≥ 3
```

(d) LEAN infoview after using *use.*

Figure 21: We use the number 4 to prove this statement.

There is no specific reason to choose four, one can choose any real number greater or equal to three to prove this.

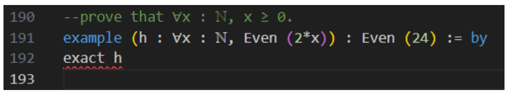
specialize tactic

Definition 2.2.68. On the contrary, when one has a ∀ quantifier in the hypothesis, one wants to use the **specialize** tactic to use a certain value for the variable.

This also corresponds to a deduction rule, namely the ∀-elimination rule:

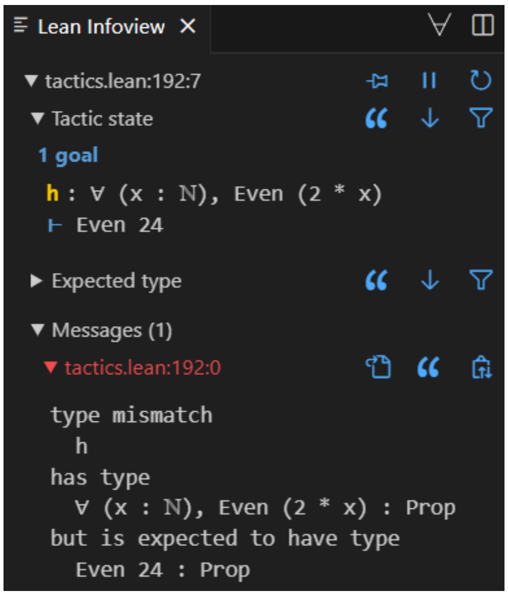
$$\frac{\forall x \in A, P(x) \qquad N \in A}{P(N)} \; (\forall\text{-elim})$$

Example 2.2.69.



```
--prove that ∀x : ℕ, x ≥ 0.
example (h : ∀x : ℕ, Even (2*x)) : Even (24) := by
exact h
```

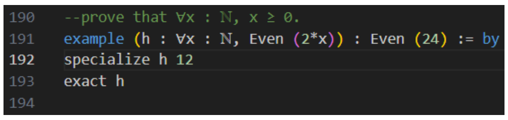
(a) Before we use *specialize.*



```
1 goal
h : ∀ (x : ℕ), Even (2 * x)
⊢ Even 24
```

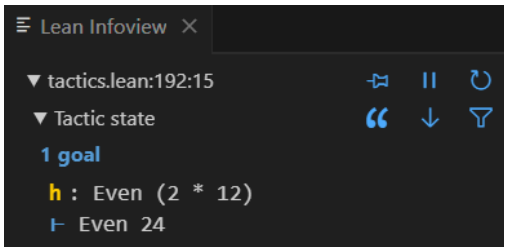
```
type mismatch
  h
has type
  ∀ (x : ℕ), Even (2 * x) : Prop
but is expected to have type
  Even 24 : Prop
```

(b) LEAN infoview before using *specialize.*



```
--prove that ∀x : ℕ, x ≥ 0.
example (h : ∀x : ℕ, Even (2*x)) : Even (24) := by
specialize h 12
exact h
```

(c) *specialize* used in VS-Code.



```
1 goal
h : Even (2 * 12)
⊢ Even 24
```

(d) LEAN infoview after using *specialize.*

Figure 22: We specialize the hypothesis to prove the desired result.

Only after specializing the hypothesis with $x = 12$, we could exact it to prove the goal.

conv_rhs and conv_lhs tactics

Definition 2.2.70. We may want to change something in the goal or a hypothesis but we just want to change one side of an equality. Then we can use the tactics **conv_rhs** => or **conv_lhs** =>.

Example 2.2.71.

```
--Show that Σ_{k=1}^{n+1} (2*k-1) = n^2.
example : Σ k in Finset.Ico 1 (n+1), (2*k-1) = n^2 := by --try it
    yourself
induction' n with d hd
simp
rw [sum_Ico_succ_top]
rw [hd]
rw [succ_eq_add_one]
ring_nf
rw [add_comm]
rw [add_left_inj]
rw [add_comm]
conv_rhs => rw [add_comm]
linarith
done
```

shortly before we finish the goal, we have to use commutativity of addition. The problem is that LEAN always goes from left to right and tries to apply the theorems. This is why we use *conv_ rhs* => here.

push_cast tactic

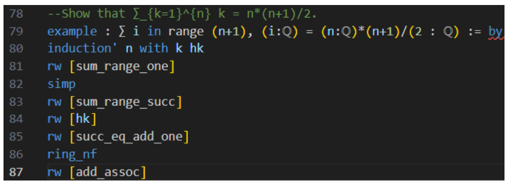
One problem we have with type theory is that 2:`nat` is not the same as 2:`int`. Now if we want to subtract something in LEAN while we have an expression of type `nat`, we have to prove that we still get a positive result, even if it is clear to us. The LEAN community developed a tactic that allows us to switch between types to overcome this problem.

Definition 2.2.72. The **push_cast** tactic can "push" the type of an expression to another one to help us solve a proof. It is an algorithmic tactic and quite technical.

Remark 2.2.73. The LEAN community was planning on integrating the *push_ cast* tactic into the simp tactic but decided to let it be its own tactic later.

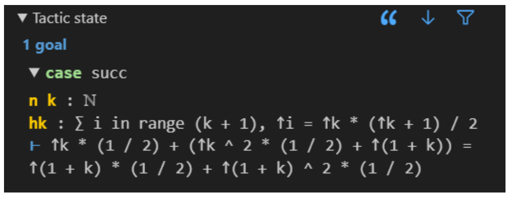
Remark 2.2.74. The *push_ cast* tactic is not easy to understand. We need to dig deep into type theory to fully understand it. Since we only use it once in all the LEAN exercise sheets, it is mentioned here but not discussed in further detail.

Example 2.2.75. *push_ cast* is used here, to prove that $\sum_{i=0}^{n} i = (n \cdot (n+1))/2$. Since n is a natural number and division is not defined for natural numbers, we need to somehow change the type of the variables.



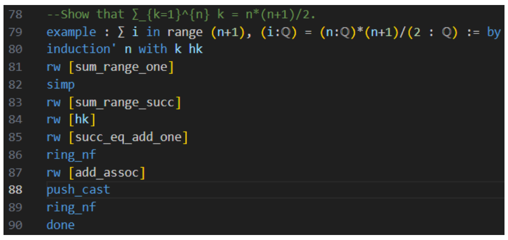
```
--Show that ∑_{k=1}^{n} k = n*(n+1)/2.
example : ∑ i in range (n+1), (i:Q) = (n:Q)*(n+1)/(2 : Q) := by
induction' n with k hk
rw [sum_range_one]
simp
rw [sum_range_succ]
rw [hk]
rw [succ_eq_add_one]
ring_nf
rw [add_assoc]
```

(a) before we used *push_ cast*.



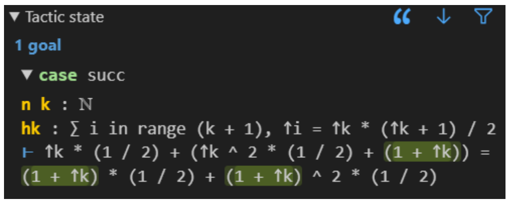
```
▼ Tactic state
1 goal
 ▼ case succ
 n k : N
 hk : ∑ i in range (k + 1), ↑i = ↑k * (↑k + 1) / 2
 ⊢ ↑k * (1 / 2) + (↑k ^ 2 * (1 / 2) + ↑(1 + k)) =
 ↑(1 + k) * (1 / 2) + ↑(1 + k) ^ 2 * (1 / 2)
```

(b) LEAN infoview before using *push_ cast*.



```
--Show that ∑_{k=1}^{n} k = n*(n+1)/2.
example : ∑ i in range (n+1), (i:Q) = (n:Q)*(n+1)/(2 : Q) := by
induction' n with k hk
rw [sum_range_one]
simp
rw [sum_range_succ]
rw [hk]
rw [succ_eq_add_one]
ring_nf
rw [add_assoc]
push_cast
ring_nf
done
```

(c) *push_ cast* used in VS-Code.



```
▼ Tactic state
1 goal
 ▼ case succ
 n k : N
 hk : ∑ i in range (k + 1), ↑i = ↑k * (↑k + 1) / 2
 ⊢ ↑k * (1 / 2) + (↑k ^ 2 * (1 / 2) + (1 + ↑k)) =
 (1 + ↑k) * (1 / 2) + (1 + ↑k) ^ 2 * (1 / 2)
```

(d) LEAN infoview after using *push_ cast*.

Figure 23: Using the *push_ cast* tactic to "push" k from $\mathbb{N}$ to $\mathbb{Q}$.

We see that from Subfigure (b) to (d), the arrow before the brackets goes before the k. Now LEAN thinks that k is a rational number, even though it would actually be a natural number. This allows us to use *ring_ nf* to finish the proof.

exfalso tactic

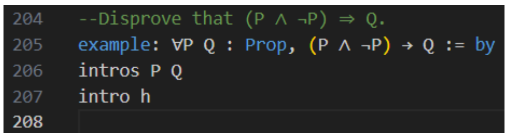
Whenever we have a hypothesis that is a contradiction, we can simply change the goal to $False$ and finish our proof.

Definition 2.2.76. The tactic **exfalso** changes the goal to $False$. This can be useful when we have a hypothesis that is a contradiction.

exfalso has another nice property. From $False$ ($\perp$), we can derive anything. In natural deduction, this is called the $\perp$-elimination rule.

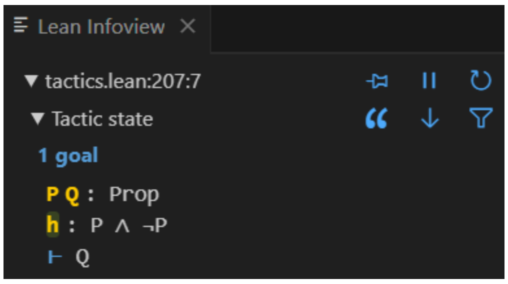
$$\frac{\perp}{P}\ (\perp\text{-elim})$$

Example 2.2.77.



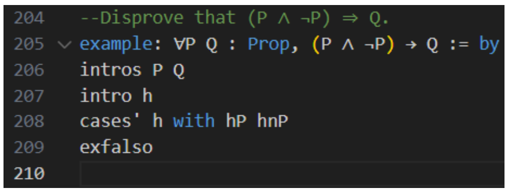
```
--Disprove that (P ∧ ¬P) ⇒ Q.
example: ∀P Q : Prop, (P ∧ ¬P) → Q := by
intros P Q
intro h
```

(a) Before we use *exfalso*.



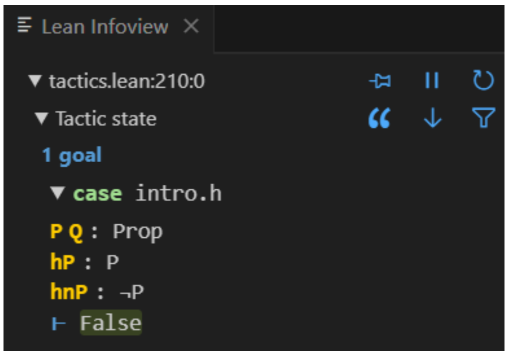
```
Lean Infoview
▼ tactics.lean:207:7
▼ Tactic state
1 goal
P Q : Prop
h : P ∧ ¬P
⊢ Q
```

(b) LEAN infoview before using *exfalso*.



```
--Disprove that (P ∧ ¬P) ⇒ Q.
example: ∀P Q : Prop, (P ∧ ¬P) → Q := by
intros P Q
intro h
cases' h with hP hnP
exfalso
```

(c) *exfalso* used in VS-Code.



```
Lean Infoview
▼ tactics.lean:210:0
▼ Tactic state
1 goal
  ▼ case intro.h
  P Q : Prop
  hP : P
  hnP : ¬P
  ⊢ False
```

(d) LEAN infoview after using *exfalso*.

Figure 24: The hypothesis makes no sense, so we change the goal to $False$.

This is an example that actually uses one further derivation rule, namely the ¬-elimination rule, which states that having P and $\neg P$, we can derive absurdity ($\perp$):

$$\frac{P \qquad \neg P}{\perp}\ (\neg\text{-elim})$$

Using constructive logic, we can prove this statement like this.

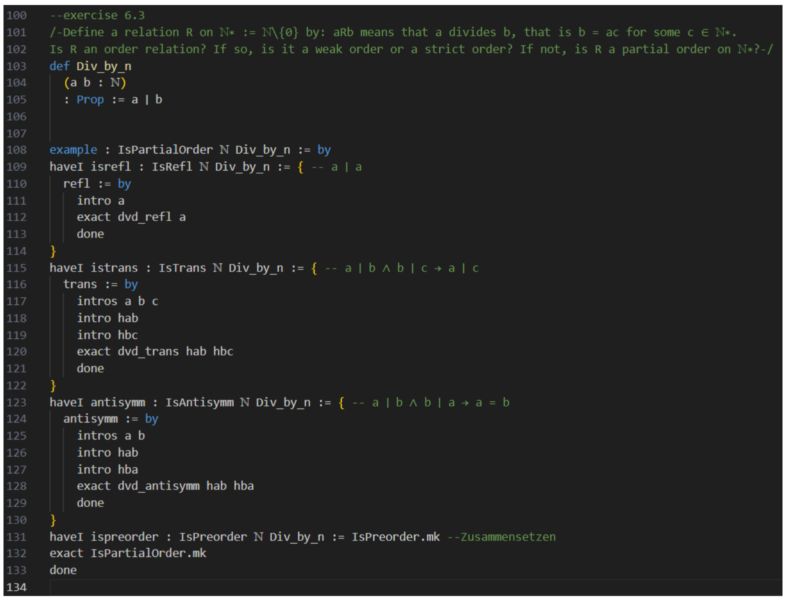
```
example : P  → (¬P) →  Q :=
  fun p : P =>
  fun np : ¬P =>
  False.elim (np p)
```

Here the $False.elim$ rule is enough to solve the proof. LEAN seems to apply the $\perp$-elimination rule automatically.

haveI tactic

Definition 2.2.78. similar to the *have* tactic, there is the **haveI** tactic that can give instances to objects like relations. Again, we first have to prove that this instance is valid before we can use it.

Example 2.2.79.



```
--exercise 6.3
/-Define a relation R on ℕ∗ := ℕ\{0} by: aRb means that a divides b, that is b = ac for some c ∈ ℕ∗.
Is R an order relation? If so, is it a weak order or a strict order? If not, is R a partial order on ℕ∗?-/
def Div_by_n
  (a b : ℕ)
  : Prop := a ∣ b


example : IsPartialOrder ℕ Div_by_n := by
haveI isrefl : IsRefl ℕ Div_by_n := { -- a ∣ a
  refl := by
    intro a
    exact dvd_refl a
    done
}
haveI istrans : IsTrans ℕ Div_by_n := { -- a ∣ b ∧ b ∣ c → a ∣ c
  trans := by
    intros a b c
    intro hab
    intro hbc
    exact dvd_trans hab hbc
    done
}
haveI antisymm : IsAntisymm ℕ Div_by_n := { -- a ∣ b ∧ b ∣ a → a = b
  antisymm := by
    intros a b
    intro hab
    intro hba
    exact dvd_antisymm hab hba
    done
}
haveI ispreorder : IsPreorder ℕ Div_by_n := IsPreorder.mk --Zusammensetzen
exact IsPartialOrder.mk
done
```

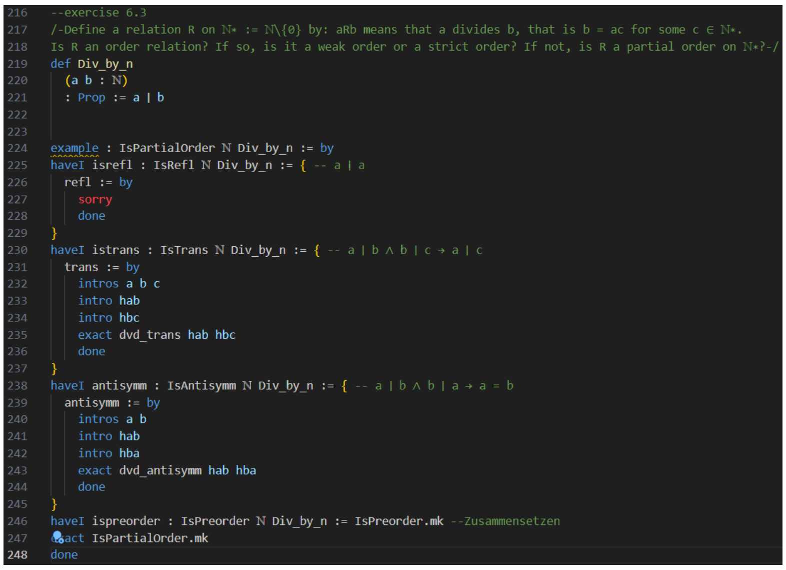
Figure 25: We can use *haveI* to build instances that we can use later.

sorry tactic

If we do not know a part of a proof right now, we can skip this step and come back to it later.

Definition 2.2.80. In LEAN, we can skip parts of the proof by typing **sorry**. Keep in mind to prove these parts at a later time.

Example 2.2.81. We can have a look at the same proof as before but leave out the proof for reflexivity.



```
--exercise 6.3
/-Define a relation R on ℕ* := ℕ\{0} by: aRb means that a divides b, that is b = ac for some c ∈ ℕ*.
Is R an order relation? If so, is it a weak order or a strict order? If not, is R a partial order on ℕ*?-/
def Div_by_n
  (a b : ℕ)
  : Prop := a ∣ b


example : IsPartialOrder ℕ Div_by_n := by
haveI isrefl : IsRefl ℕ Div_by_n := { -- a ∣ a
  refl := by
    sorry
    done
}
haveI istrans : IsTrans ℕ Div_by_n := { -- a ∣ b ∧ b ∣ c → a ∣ c
  trans := by
    intros a b c
    intro hab
    intro hbc
    exact dvd_trans hab hbc
    done
}
haveI antisymm : IsAntisymm ℕ Div_by_n := { -- a ∣ b ∧ b ∣ a → a = b
  antisymm := by
    intros a b
    intro hab
    intro hba
    exact dvd_antisymm hab hba
    done
}
haveI ispreorder : IsPreorder ℕ Div_by_n := IsPreorder.mk --Zusammensetzen
exact IsPartialOrder.mk
done
```

Figure 26: Using *sorry*, we can omit the proof for reflexivity.

Obviously, one would need to prove the missing part too to finish the proof. This is why at the start of the exercise, where “example” is written, we have a yellow underline showing us that the proof has not been concluded yet. It still is a useful tactic as we never get stuck in parts of proofs.

There are countless important theorems to use in LEAN. We have summarized the most important addition, subtraction, multiplication and division theorems in a “cheat sheet” that can be found on my Github [Bot23]. But we will not list them all here since they would take up too much space.

After seeing so many tactics that we can use in LEAN, and knowing how we can build a LEAN project, it remains to state how to best start learning LEAN.

Natural Number Game

The Natural Number Game (NNG) was originally developed by Kevin Buzzard and Mohammad Pedramfar in 2019 [BP21]. At that time, it was written for LEAN 3. The reason we write about this game is because it was my first entrance to LEAN and it had a great influence on my levels.

With the NNG one can introduce the natural numbers in LEAN step-by-step starting with Peano's axioms and proving all the addition, multiplication and power rules as well as the ordering on the natural numbers. The game also gives one first insights into propositional logic and functions. It is made up of 10 worlds with 4 to 17 levels each covering the different topics. This gave me the idea to structure my teachings in levels with exercises, i.e. my levels correspond to the worlds of NNG and my exercises to the levels.

When we started to teach LEAN for this thesis, the LEAN community was just changing from LEAN 3 to LEAN 4. We already planned on using the NNG for the teachings when we would come to the natural numbers and were a bit concerned that we would need to make the NNG in LEAN 3 after learning LEAN 4 with the students. Fortunately, a working version of NNG for LEAN 4 was finished by the time the students arrived at the natural numbers in their course [BE23].

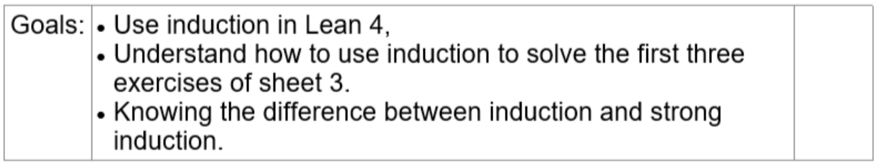
We highly recommend starting with the NNG when trying to learn LEAN. It gives a perfect first impression of what proving mathematical statements with LEAN looks like.

3 Methods

In this section, we consider the teaching methods and data gathering. The teaching methods were used to prepare weekly sessions we had with five volunteer first-year mathematics students.

3.1 Teaching Methods

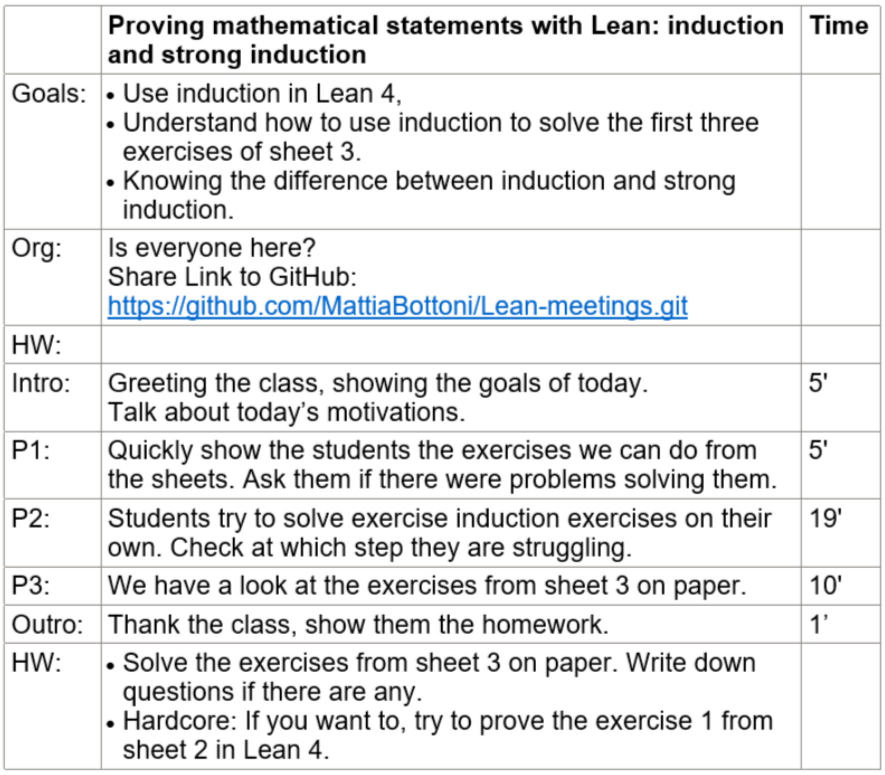
One of the most important things to do in meetings with students is to show them the goals for each session at the beginning. This helps the students to focus on the essential topics and the teacher to reach his pedagogical goals. In [FF10] it is also proven to heighten students' achievements. In the meetings the goals were written following Mager's structure [Mag78]. Meaning that they would not only contain a content but also a skill the students should acquire. In Figure 27 below, one can see an example from one of the meetings.



Goals:	• Use induction in Lean 4, • Understand how to use induction to solve the first three exercises of sheet 3. • Knowing the difference between induction and strong induction.	

Figure 27: The goals from our 4th meeting.

To prepare the sessions, the same table explained in Figure 28 below was always used. Carefully preparing the lessons is the key to successful teaching, especially for young teachers [Mey20].



	Proving mathematical statements with Lean: induction and strong induction	**Time**
Goals:	• Use induction in Lean 4, • Understand how to use induction to solve the first three exercises of sheet 3. • Knowing the difference between induction and strong induction.	
Org:	Is everyone here? Share Link to GitHub: https://github.com/MattiaBottoni/Lean-meetings.git	
HW:		
Intro:	Greeting the class, showing the goals of today. Talk about today's motivations.	5'
P1:	Quickly show the students the exercises we can do from the sheets. Ask them if there were problems solving them.	5'
P2:	Students try to solve exercise induction exercises on their own. Check at which step they are struggling.	19'
P3:	We have a look at the exercises from sheet 3 on paper.	10'
Outro:	Thank the class, show them the homework.	1'
HW:	• Solve the exercises from sheet 3 on paper. Write down questions if there are any. • Hardcore: If you want to, try to prove the exercise 1 from sheet 2 in Lean 4.	

Figure 28: Preparation table for the 4th meeting.

This table is something picked up during substitute teaching in a high school. As one can see in the table, directly after the goals, we showed the students some motivations for why we are learning today's topics. Giving them motivation and not only goals is essential for successful and efficient learning [EW02]. Note that the fourth row (HW) is empty because in this meeting, we did not discuss the homework from the week before.

We differ between extrinsic and intrinsic motivation. The first one comes from goals and expectations society has on a person, while the second one comes from within and is fueled by individual interests and curiosities. Deci and Ryan argued in [DR93] that extrinsic and intrinsic motivation are in fact not antagonists, but that one can turn extrinsic motivation into intrinsic motivation. This process consists of four stages, called: external regulation, introjected regulation, identified regulation and integrated regulation. It describes how a motivation given from outside, slowly becomes internalised until the student feels self-determined and experiences intrinsic motivation. With the questionnaire in Section 3.2 we conduct, we want to find out the motivation of the LEAN students to see if higher motivation leads to better learning results.

The meetings were structured in the sandwich method [Wah13]. This method relies on a varied course of the lesson, meaning that one changes a lot between frontal teaching and experimenting parts. We were taught this lesson structure in the subject didactics course at UZH. It gives the students the possibility to work for themselves for a great part of the lesson.

When choosing the exercises we wanted to discuss with them, it is very important to have an appropriate difficulty level. If the content is too difficult, students are overwhelmed and frustrated if the content is too easy, students get bored [Pet22]. As a teacher, one may try to reach the so-called "zone of proximal development" [VC78]. It describes the difficulty level which is exactly a bit higher than the knowledge students have, but not so high that they cannot reach it. This is one of the reasons for the restructuring of the meetings in the last few weeks of the semester, as elaborated in Chapter 5.2.

Heterogeneity is a frequently talked about topic in teaching [Wod14]. In each class, one encounters a lot of different skill levels. While it would be the best to treat each student individually, this is often not possible due to the big class sizes. Since we did not have many students, we managed to treat the students according to their needs, a privilege that is not often encountered. With Ladina, we had a student who already knew very much about LEAN compared to the others and even to us. Fortunately, having the extra meetings with her alone, allowed us to have differentiated teaching [HS21], meaning that we could raise the difficulty level for her in particular.

That we need to adapt our teachings to the coming of new digital media should be obvious. However, we are at a very slow change of heart and lack good tools to teach subjects with digital tools [Pet14]. Admittedly, while digital technology has been here for almost 40 years now, it takes time to research how to use this technology well in classrooms. Some tendencies and methods are starting to arise, and others are underlining the importance of digital teaching. We cannot allow our schools to undergo zero changes while the world around us spins faster each day. That is why some research papers call for media-aware teaching, where we need to develop critical thinking towards information and the skill to deal with the flow of information around us [HA+16]. Others warn that teaching with digital media does not substitute good teaching, it should just evolve it [FBC17].

In our case, we use the mix of both methods explained above. Teaching the students digital affinity by expanding the teaching with digital resources, without losing its meaning. However, we are certainly not alone in this. The term *STEM* teaching (Science, Technology, Engineering and Mathematics) has been heard a lot in the last years. [HA+16] and [FBC17] talk about the importance of integrating various subjects in teaching, and LEAN does that very well, at least with technology and mathematics. In summary, educational institutions need to evolve their teachings to suffice the requirements of the digital world. Students need to have the chance to learn information literacy, critical thinking towards information, creative work with digital media, and interdisciplinary STEM teaching. LEAN gives us the possibility to do that, and it is important to understand that this is not self-evident.

Finally, the class climate is essential for good learning and teaching. We had a very informal and eye-to-eye relationship during our sessions. This helped build up a strong connection with the students, which would later help us with the interviews since most of them agreed to participate. It is important that a good climate does not lead to lesser learning [CMP14]. After all, they were still students and not colleagues. To provide this I would always sit next to a screen with the content on it, while the students would build a half-circle around me. This secured a closed but still distant relationship. Next to better learning, a good class climate can also lead to an increase in self-confidence, willingness to perform, positive attitude toward lessons and the school, social behaviour and the formation of interest [Mey21]. The latter helps again to encourage intrinsic motivation.

3.2 Data Gathering

The main goal of this thesis is to observe the effect that both teaching LEAN and teaching with LEAN has on one's proving skills. To do so, we need to find a way to measure the quality of proof, and design suitable interviews and questionnaires for LEAN and Non-LEAN students.

Proof Structure

Thoma and Iannone explain different points one can check to observe the quality of a proof [TI21], and here we grade students' performance by these criteria. The grading tables can be found in Interviews and Questionnaire, and the results are stated in Chapter 4.

- **Definitions and their use**: With this, we check if students understand the definitions given and if they can apply them correctly in the proof.
- **Mathematical symbols and their use**: Mathematical symbols are omnipresent in mathematical proofs, understanding and applying them correctly is necessary to provide a proof.
- **Logical statements and their links**: Students need to understand how to deal with implications or equivalences. Dealing correctly with quantifiers also falls into this criterion.
- **High level ideas**: Many proofs require this steep step, where one gets the key idea with how one wants to go. Here we also consider if students find links between mathematical theories such as set theory and first-order logic. While doing grading in the criteria logic and definitions, we consider these skills of students.
- **Modular structure of the proof**: This criterion checks the ability to split up a proof into different sub-proofs and to put everything together at the end. It is really about the skeleton of the proof.
- **Use of examples**: Students can use examples to find a pattern in the proof that might help them finish it.

Interviews and Questionnaire

For this thesis, we interview both groups, LEAN and Non-LEAN students, in a quantitative test. To better understand the amount of time spent with LEAN, we additionally prepared a questionnaire only for the LEAN students. In this section, we present and justify the questions chosen for the interview and we discuss the main focuses for each as well as the grading tables used to grade the interviews related to the grading criteria explained above. Then we present the questionnaire. The results for the interviews and the evaluation of the questionnaire can be found in Chapter 4.

The four interview questions are the following.

1. *Prove the following deMorgan law without a truth table:*

   $$\neg(p \vee q) = (\neg p) \wedge (\neg q)$$

   With this question we would like to check if LEAN students developed additional skills to deal with propositional logic proofs and the way that Non-LEAN students approach these kinds of proofs.

2. *Using all axioms of addition and multiplication of natural numbers, prove:*

   $$\forall a, b, c \in \mathbb{N},\ a \cdot (b + c) = a \cdot b + a \cdot c$$

   In this question, we want to measure the understanding of proof by induction and the definition of the successor function. We also check their correct handling with propositional equalities in the base case.

3. *Recall that:*

   $a \equiv b \mod n \Leftrightarrow (\exists c, d, r \in \mathbb{N},\ a = c{\cdot}n{+}r \text{ and } b = d{\cdot}n{+}r) \Leftrightarrow (\exists k \in \mathbb{Z}, a{-}b = k{\cdot}n)$.

   Show that congruence mod n *with* $n = 3$ *is an equivalence relation.*
   This question aims to see if students can apply the proof of an equivalence relation in the context of modulo. We also check their understanding of the definition of modulo.

4. *If* $J \neq \emptyset$ *and* $J \subseteq I$, *does it follow that* $\bigcup_{\alpha \in J} A_\alpha \subseteq \bigcup_{\alpha \in I} A_\alpha$? *What about* $\bigcap_{\alpha \in J} A_\alpha \subseteq \bigcap_{\alpha \in I} A_\alpha$?

   For this last question, we want to know more about students' understanding about set-theoretic proofs and the definitions of some more complex set structures.

The idea of the interview is to cover as many topics from the lecture as possible. We see that there is a question about propositional logic, one about the natural numbers and natural induction, one about the definition of an equivalence relation and one on sets, namely on indexed sets and operations on sets. The questions in the interviews can also be shown to be relevant, as similar topics were covered in the exercise sheets, the final exam and even in [TI21], as one can see in the table below.

	Propositional Logic	**Basic Set Theory**	**Proofs**	**Mathematical Induction**	**Relations and Functions**	**Cardinality**	**Natural Numbers**	**Order Relations**
Thoma and Iannone			2	1	2		1	
Interview	1	1		1	1		1	
Exercise Sheets	2	2	1	3	2	1	2	1
Exam	1		2	1	2	1		1

Table 7: Overview of the topics asked about in various examinations.

Table 7 demonstrates how often a topic is covered in either the Thoma and Iannone paper [TI21], the interviews we made, the exercise sheets of the course “Foundations of Mathematics” and the final exam of that course. One can see that there is a high overlap between the topics chosen.

Question 1 is specially conceived to give the LEAN students an advantage and Exercise 4 is well suited for the students who did not learn LEAN. This is because Non-LEAN students have never seen how to prove propositional logic without truth tables and we have not spent much time on exercises like Question 4 with the LEAN students. In these four questions, apart from the focuses mentioned above, we check their proof writing based on the characteristics explained in Section 3.2. The following tables show the grading criteria that are used to grade the exercises of the students.

Question 1			
	0 Points	1 Point	2 Points
D	Needs help with the definitions, even when given.	Understands the needed definitions, i.e. “or”/“and”, but maybe does not know them all.	Uses them in a confident way.
MS	Struggles with writing mathematics.	Has a clean way of writing mathematics.	No neglection detected.
L	Makes confusions with the “not” operator.	Captures the meaning of the “not” operator, but needed help/made mistakes.	Applies all logical connectives correctly and without help.
HLI	Does not split the proof in two parts.	Splits the proof in two parts, maybe with a bit help.	Sees how to use the “and” in the proof.
S	Bad approach of proof, little to no structure visible.	Basic structure detected, maybe some parts not formulated well enough.	Clean and confident structure.
UE	Never used an example.	Talks about examples.	Uses examples in a helpful way.

Table 8: Grading for Question 1.

Question 2			
	0 Points	1 Point	2 Points
D	Needs help with the definitions, even when given.	Understands the needed definitions, i.e. some axioms, but maybe does not know them all.	Uses them in a confident way.
MS	Struggles with writing mathematics.	Has a clean way of writing mathematics.	No neglection detected.
L	Does not capture the meaning of “for all’.	Captures the meaning of “for all”, but needed help/made mistakes.	Applies all logical connectives correctly and without help.
HLI	Needs to be told to use induction and did not manage to apply mul_succ.	Understands to use induction, maybe with a bit help.	Sees how to use mul_succ.
S	Bad approach of proof, little to no structure visible.	Basic structure detected, maybe some parts not formulated well enough.	Clean and confident structure.
UE	Never used an example.	Talks about examples.	Uses examples in a helpful way.

Table 9: Grading for Question 2.

Question 3			
	0 Points	1 Point	2 Points
D	Needs help with the definitions, even when given.	Understands the needed definitions, i.e. refl, symm, trans, but maybe does not know them all.	Uses them in a confident way.
MS	Struggles with writing mathematics.	Has a clean way of writing mathematics.	No neglection detected.
L	Struggles with the "if-and-only-ifs'.	Captures the meaning of "if-and-only-ifs', but needed help/made mistakes.	Applies all logical connectives correctly and without help.
HLI	Does not know how to prove symm and trans.	Understands how to prove symm and trans, maybe with a bit help.	Sees how to prove symm and trans.
S	Bad approach of proof, little to no structure visible.	Basic structure detected, maybe some parts not formulated well enough.	Clean and confident structure.
UE	Never used an example.	Talks about examples.	Uses examples in a helpful way.

Table 10: Grading for Question 3.

Question 4			
	0 Points	1 Point	2 Points
D	Needs help with the definitions, even when given.	Understands the needed definitions, i.e. union and intersection, but maybe does not know them all.	Uses them in a confident way.
MS	Struggles with writing mathematics.	Has a clean way of writing mathematics.	No neglection detected.
L	Does not capture the "does-it-follow' correctly.	Captures the meaning of "how-it-follows', but needed help/made mistakes.	Follows through correctly and without help.
HLI	Does not set x in union over J.	Understands how to prove the subset, maybe with a bit help.	Sees how to use the definition of union and intersection.
S	Bad approach of proof, little to no structure visible.	Basic structure detected, maybe some parts not formulated well enough.	Clean and confident structure.
UE	Never used an example.	Talks about examples.	Uses examples in a helpful way.

Table 11: Grading for Question 4.

These grading tables are used as support for the marking of the interviews. Note that after individual gradings, we had intense discussions about the scores given. During these discussions, some criteria of the grading tables might have been stretched a bit. This would, for example, be the case if a student did not write down a definition correctly, but it was clear in the interview that he or she had understood the definition well enough.

The questionnaire contains questions that give us insight into how LEAN affected the students who worked with us during the semester. The answers are collected using a Google form. Students are asked to answer the following questions.

- Did LEAN motivate you to spend more time on a proof than usual? That is, would you spend more time trying to solve an exercise in LEAN or in the Natural Number Game than on paper?
- Did LEAN improve your proving skills in general?
- Did LEAN influence how well you felt prepared for the final exam?
- How often did you use LEAN besides the meetings?
- Should students be taught mathematics with LEAN in the future?
- Open question: Do you plan to continue working with LEAN during your studies? Why or why not?
- Open question: Would you have done something differently in the way we held our meetings with you?

The first few questions could be answered with “No/Never”, “Rarely/Hardly”, “Sometimes/A bit” or “Yes/Often”. Question 5 has the options “No”, “No opinion”, “That depends” or “Yes”. All non-open questions have the optional possibility to leave a comment. This questionnaire helps us to better analyze the students’ progress. If a LEAN student does not perform so well in the exam and/or the interview, we check how these students answer the questionnaire. If they answer Question 3 with “Never” or “Rarely”, we know that the problem is not the meetings themselves, but the lack of motivation to work with LEAN at home. The results of this questionnaire can be found in Section 4.

4 Interview Results

Based on [TI21], we first compare the performance of the students participating in the interviews during the first exercise sheet and the exam, to make sure that LEAN students are not already better students before the sessions start. Then, the results from the interviews are shown and commented on. Finally, we discuss some individual students or compare LEAN and Non-LEAN students with similar performance and then present the significance of the results by considering an independent t-test and a Mann-Whitney U-test.

Exercise Sheets and the Exam

In this section, we present the means and median scores from the exercise sheets and the final exam of the course "Foundations of Mathematics". We do this to compare the performance between LEAN and Non-LEAN students. Contrary to the results of the interviews, this data contains the scores from all Non-LEAN students of the course.

	Sheet 1	**Sheet 2**	**Sheet 3**	**Sheet 4**	**Sheet 5**	**Sheet 6**	**Exam**
LEAN							
Median	19	16.5	20	20	18.5	17.3	47
Mean	18.7	17.3	19.8	19.9	19.1	17.6	46
Non-LEAN							
Median	19	18.5	19	18.5	17.5	18	35
Mean	18.2	17.4	18.7	17.7	16.2	17.2	34.9
Total possible	20	20	20	20	20	20.0	60

Table 12: LEAN and Non-LEAN students exercise sheets and exam performance.

We have access to the anonymized scores of all the students, where just the LEAN students are labelled as **L**. Since not all students handed in all the exercise sheets, we always delete this data and just compute the mean and the median of the students that have handed in the exercise sheets. That way, we consider around 35 Non-LEAN and five LEAN students. For the exam, we consider the final score, but not the mark of the exam. That way, we do not have to consider the grading key used. One can see that while in the first exercise sheet, LEAN and Non-LEAN students performed fairly equally, in the exam, LEAN students scored over 10 points more on average than Non-LEAN students.

Interviews

We start by stating the mean, lowest and highest score (points for all six criteria summed) of each question, without comparing LEAN and Non-LEAN students.

Question	**Mean Score**	**Lowest Score**	**Highest Score**
1	4.4	0	9
2	6.4	2	10
3	8.1	4	10
4	7.7	3	12

Table 13: Mean, lowest and highest score for each exercise of the interview.

Keep in mind that having six criteria, the highest possible score would be 12 points per exercise. We talk about the results for each question more in detail later.

Before we discuss the results of each question, here are the results of the total scores from the interviews:

	Laurin	**Nevia**	**Ladina**	**Niculin**	**Lavinia**	**Nuot**	**Linard**	**Nicola**	**Liun**	**LEAN Mean**	**Non-LEAN Mean**	**Mean**
Definitions	4	3	8	4	6	2	7	8	4	5.8	4.25	5.1
Mathematical Symbols	3	6	7	7	8	4	7	6	5	6	5.75	5.9
Logic	4	6	8	5	7	2	6	7	4	5.8	5	5.4
High Level Idea	4	5	8	4	7	1	4	6	2	5	4	4.6
Structure of the Proof	4	4	7	3	6	2	4	7	4	5	4	4.6
Use of Examples	0	0	1	2	2	0	2	2	1	1.2	1	1.1
Total	19	24	39	25	36	11	30	36	20	28.8	24	26.7

Table 14: Total scores in each criterion for all students.

General Mean	**Laurin**	**Nevia**	**Ladina**	**Niculin**	**Lavinia**	**Nuot**	**Linard**	**Nicola**	**Liun**	**LEAN Mean**	**Non-LEAN Mean**	**Mean**
Definitions	1	0.75	2	1	1.5	0.5	2	1.5	1	1.45	1.06	1.3
Mathematical Symbols	0.75	1.5	1.75	1.75	2	1	1.75	1.5	1.25	1.5	1.44	1.5
Logic	1	1.5	2	1.25	1.75	0.5	1.5	1.75	1	1.45	1.25	1.4
High Level Idea	1	1.25	2	1	1.75	0.25	1	1.5	0.5	1.25	1	1.1
Structure of the Proof	1	1	1.75	0.75	1.5	0.5	1	1.75	1	1.25	1	1.1
Use of Examples	0	0	0.25	0.5	0.5	0	0.5	0.5	0.25	0.3	0.25	0.3
Total Mean	4.75	6	9.75	6.25	9	2.75	7.5	9	5	7.2	6	6.7

Table 15: Mean scores in each criterion for all the students.

In Table 14 we see the total scores for each criterion and the total score for the interview. We can see that the mean score for LEAN students is higher in each criterion. The highest difference in average is for the *Definitions* criterion, followed by the *Logic* criterion. This could be due to the clean and precise structure LEAN asks one to use.

Table 15 needs some clarifications. We see the mean scores for each criterion over all four exercises. For example, Nevia scored 1.5 points on average in the logic criterion over all four exercises. In the last row, we add the average points of each criterion summed, which is the same as considering the mean score for each question at the interview. So Tables 14 and 15 do not give us different results, they just represent them in different ways.

From these tables one can already see that LEAN students performed slightly better than Non-LEAN students. We now have a look at each question in detail, to see if this is the case for each question.

Question 1

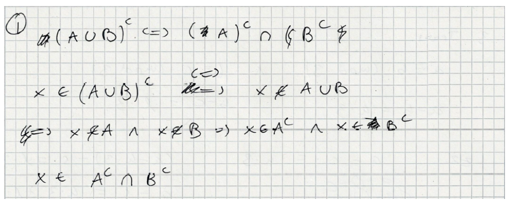
Prove the following deMorgan law without a truth table:

$$\neg(p \vee q) = (\neg p) \wedge (\neg q)$$

Question 1	Laurin	Nevia	Ladina	Niculin	Lavinia	Nuot	Linard	Nicola	Liun	**Averarge LEAN**	**Mean Non-LEAN**	**Mean**
D	0	0	2	0	0	0	2	2	1	1	0.5	0.8
MS	0	2	2	2	2	0	1	1	1	1.2	1.25	1.2
L	0	1	2	1	1	0	1	2	1	1	1	1
HLI	0	1	2	0	1	0	1	1	1	1	0.5	0.8
S	0	0	1	0	0	0	1	2	0	0.4	0.5	0.4
UE	0	0	0	1	0	0	1	0	0	0.2	0.25	0.2
Total	0	4	9	4	4	0	7	8	4	4.8	4	4.4

Table 16: Results from Question 1.

While Nicola manages to get eight points, he solves Question 1 using sets. He is not the only one trying that approach. Linard tries using sets to prove the statement too. While it is certainly not the same setting, one can argue that by the PAT interpretation, an element of a set corresponds to a proof for a proposition, meaning that we can prove DeMorgan's law in an equivalent way using set theory. Ladina is the only one able to solve this the intended way, just see Figure 29 below for Ladina's and Nicola's solution. It turns out that solving Question 1 without a truth table is too tough. Many make the mistake that they want to use the statement to prove the statement. We can see that this mistake happens in a few of the interviews. Indeed, we observe that two Non-LEAN students out of four, and one LEAN student out of five make this mistake. It is interesting to observe that the LEAN student is one with a lower performance but the two Non-LEAN students that make the mistake are average performers. While LEAN certainly helps understand useful things like $\neg P \iff (P \Rightarrow False)$, proving with first-order logic is an unfamiliar notation that cannot be learned quickly. It seems that one needs to spend more time with LEAN to master the skills of propositional logic proofs.



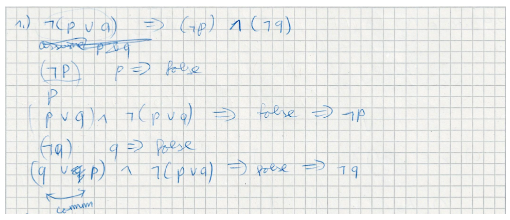
① $(A \cup B)^c \iff (A)^c \cap B^c$

$x \in (A \cup B)^c \overset{\iff}{\Rightarrow} x \notin A \cup B$

$\iff x \notin A \wedge x \notin B \Rightarrow x \in A^c \wedge x \in B^c$

$x \in A^c \cap B^c$

(a) Nicola's solution.



1.) $\neg(p \vee q) \Rightarrow (\neg p) \wedge (\neg q)$

~~assume $p \vee q$~~

$(\neg P)$ $p \Rightarrow$ false

p

$(p \vee q) \wedge \neg(p \vee q) \Rightarrow$ false $\Rightarrow \neg p$

$(\neg q)$ $q \Rightarrow$ false

$(q \vee p) \wedge \neg(p \vee q) \Rightarrow$ false $\Rightarrow \neg q$

comm

(b) Ladina's solution.

Figure 29: Two solutions for Question 1. Once with sets and once with propositional logic.

This figure shows the best LEAN and Non-LEAN performance of Question 1. As we can see, the two students approach very differently.

Question 2

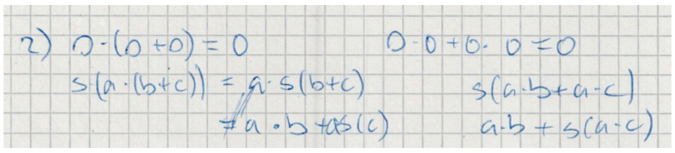
Using all axioms of addition and multiplication of natural numbers, prove:

$$\forall a, b, c \in \mathbb{N}, \ a \cdot (b + c) = a \cdot b + a \cdot c$$

Question 2	Laurin	Nevia	Ladina	Niculin	Lavinia	Nuot	Linard	Nicola	Liun	**Averarge LEAN**	**Mean Non-LEAN**	**Mean**
D	1	0	2	0	2	0	1	2	1	1.4	0.5	1
MS	1	2	1	1	2	1	2	1	2	1.6	1.25	1.4
L	1	1	2	1	2	1	2	2	2	1.8	1.25	1.6
HLI	1	1	2	1	2	0	1	2	1	1.4	1	1.2
S	2	1	2	0	2	0	1	1	2	1.8	0.5	1.2
UE	0	0	0	0	0	0	0	0	0	0	0	0
Total	6	5	9	3	10	2	7	8	8	8	4.5	6.4

Table 17: Results from Question 2.

While most of the students are able to figure out that a proof by induction is needed, some of them struggle with choosing the variable to do induction on. For instance, in the following figure, Niculin want to do induction on all three variables.

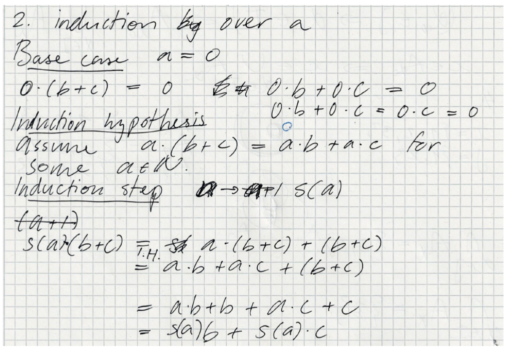

2) $0\cdot(0+0)=0$ $\quad 0\cdot 0+0\cdot 0=0$

$s(a\cdot(b+c)) = a\cdot s(b+c)$ $\quad s(a\cdot b+a\cdot c)$

$\neq a\cdot b + as(c)$ $\quad a\cdot b + s(a\cdot c)$

Figure 30: Niculin's solution for Question 2.

We check if they use the propositional equality correctly. As we see in Figures 31 and 32 below, both LEAN and Non-LEAN students are dealing with it wrongly or correctly.


2. induction ~~by~~ over a

Base case $a=0$

$0\cdot(b+c) = 0 \quad 0\cdot b+0\cdot c = 0$

$0\cdot b+0\cdot c = 0\cdot c = 0$

Induction hypothesis

assume $a\cdot(b+c) = a\cdot b+a\cdot c$ for some $a\in\mathbb{N}$.

Induction step ~~$a\to a+1$~~ $s(a)$

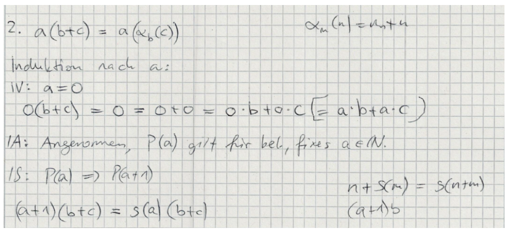
~~$(a+1)$~~

$s(a)(b+c) \overset{}{=}_{I.H.} a\cdot(b+c)+(b+c)$

$= a\cdot b+a\cdot c+(b+c)$

$= a\cdot b+b+a\cdot c+c$

$= s(a)b + s(a)\cdot c$


2. $a(b+c) = a(\alpha_b(c))$ $\quad \alpha_m(n) = m+n$

Induktion nach a:

IV: $a=0$

$0(b+c) = 0 = 0+0 = 0\cdot b+0\cdot c\ [= a\cdot b+a\cdot c)$

IA: Angenommen, $P(a)$ gilt für bel., fixes $a\in\mathbb{N}$.

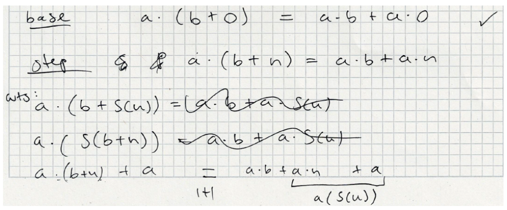
IS: $P(a) \Rightarrow P(a+1)$ $\quad n+S(m) = S(n+m)$

$(a+1)(b+c) = s(a)(b+c)$ $\quad (a+1)b$

(a) Lavinia's solution. (b) Nevia's solution.

Figure 31: The propositional equality used correctly.

While they both use the notion of propositional equality correctly, Nevia struggles more with the definition of successor multiplication than Lavinia does. Having played the Natural Number Game could have given Lavinia an advantage.

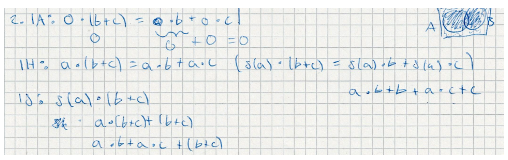

base $\quad a\cdot(b+0) = a\cdot b+a\cdot 0$ ✓

step $\quad a\cdot(b+n) = a\cdot b+a\cdot n$

wts: $a\cdot(b+S(n)) =$ ~~$(a\cdot b+a\cdot S(n))$~~

$a\cdot(S(b+n))$ ~~$= a\cdot b+a\cdot S(n)$~~

$a\cdot(b+n)+a \underset{\text{I+I}}{=} a\cdot b+\underbrace{a\cdot n+a}_{a(S(n))}$


2. IA: $\underset{0}{0\cdot(b+c)} = \underbrace{0\cdot b}_{0}+0\cdot c \quad +0 = 0$

IH: $a\cdot(b+c) = a\cdot b+a\cdot c \quad (s(a)\cdot(b+c) = s(a)\cdot b+s(a)\cdot c)$

$a\cdot b+b+a\cdot c+c$

IS: $s(a)\cdot(b+c)$

$a\cdot(b+c)+(b+c)$

$a\cdot b+a\cdot c+(b+c)$

(a) Nicola's solution. (b) Linard's solution.

Figure 32: The propositional equality used incorrectly.

In Figure 32 above we see two students who wrongly use the propositional equality. It could be that it is just a sloppy notation mistake, but such mistakes can lead to problems understanding the logic behind proving a statement.

Question 3

Recall that:

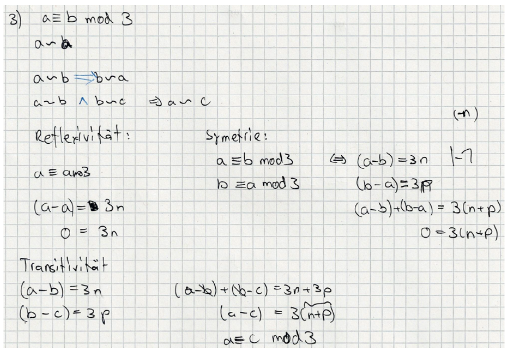
$$a \equiv b \mod n \Leftrightarrow (\exists c, d, r \in \mathbb{N},\ a = c{\cdot}n{+}r \text{ and } b = d{\cdot}n{+}r) \Leftrightarrow (\exists k \in \mathbb{Z}, a{-}b = k{\cdot}n).$$

Show that congruence mod n *with* $n = 3$ *is an equivalence relation.*

Question 3	Laurin	Nevia	Ladina	Niculin	Lavinia	Nuot	Linard	Nicola	Liun	**Averarge LEAN**	**Mean Non-LEAN**	**Mean**
D	2	2	2	2	2	1	2	2	2	2	1.75	1.9
MS	2	1	2	2	2	2	2	2	1	1.8	1.75	1.8
L	2	2	2	2	2	0	1	2	0	1.4	1.5	1.4
HLI	2	1	2	1	2	0	1	2	0	1.4	1	1.2
S	2	2	2	2	2	1	1	2	2	1.8	1.75	1.8
UE	0	0	0	0	0	0	0	0	0	0	0	0
Total	10	8	10	9	10	4	7	10	5	8.4	7.75	8.1

Table 18: Results from Question 3.

Apart from a few students, everybody can state the criteria one needs to show that something is an equivalence relation correctly. We compare two LEAN and Non-LEAN solutions for an adequate and a good proof of this statement.


3) $a \equiv b \mod 3$

$a \sim a$

$a \sim b \Rightarrow b \sim a$

$a \sim b \wedge b \sim c \Rightarrow a \sim c$

Reflexivität:

$a \equiv a$ mod 3

$(a-a) = 3n$

$0 = 3n$

Symetrie:

$a \equiv b \mod 3 \Leftrightarrow (a-b) = 3n$ | $\cdot (-1)$

$b \equiv a \mod 3$ $(b-a) = 3p$

$(a-b)+(b-a) = 3(n+p)$

$0 = 3(n+p)$

Transitivität

$(a-b) = 3n$

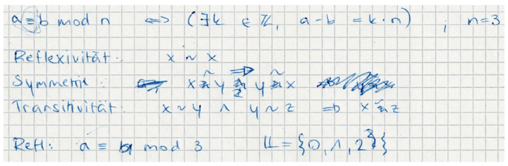
$(b-c) = 3p$

$(a-b)+(b-c) = 3n+3p$

$(a-c) = 3\overbrace{(n+p)}$

$a \equiv c \mod 3$

(a) Liun's solution.


$a \equiv b \mod n \Leftrightarrow (\exists k \in \mathbb{Z},\ a-b = k \cdot n)$; $n=3$

Reflexivität: $x \sim x$

Symmetrie: $x \sim y \Rightarrow y \sim x$

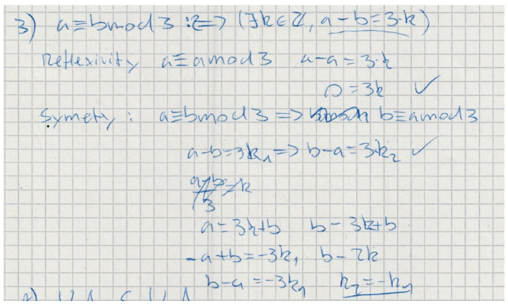
Transitivität: $x \sim y \wedge y \sim z \Rightarrow x \sim z$

Refl: $a \equiv b \mod 3$ $\mathbb{L} = \{0, 1, 2\}$

(b) Nuot's solution.

Figure 33: These two students struggle a bit with proving this statement.

Nuot more or less knows the criteria for an equivalence proof, but he is not able to prove the criteria. While Liun's proof looks a bit more elaborate, he is not able to prove symmetry correctly, as he tries to use the same method as for the transitivity.



3) $a \equiv b \bmod 3 :\Longleftrightarrow (\exists k \in \mathbb{Z}, \underline{a-b=3\cdot k})$

Reflexivity $a \equiv a \bmod 3$ $\quad a-a=3\cdot k$

$0=3k$ ✓

Symetry : $a \equiv b \bmod 3 \Rightarrow$ ~~[illegible]~~ $b \equiv a \bmod 3$

$a-b=3k_1 \Rightarrow b-a=3k_2$ ✓

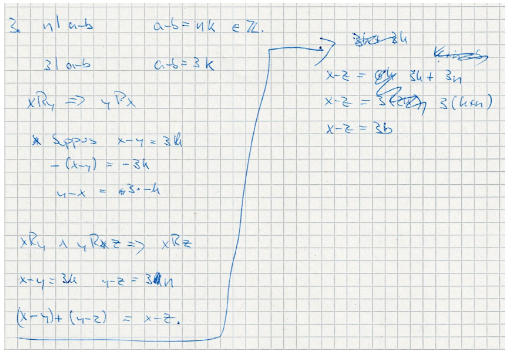
~~$\frac{a-b}{3}$~~ $\neq k$

$a=3k+b \quad b-3k+b$

$-a+b=-3k_1 \quad b-2k$

$b-a=-3k_1 \quad \underline{k_2=-k_1}$

(a) Niculin’s solution.



3. $n \mid a-b \qquad a-b=nk \in \mathbb{Z}.$

$3 \mid a-b \qquad a-b=3k$

$xRy \Rightarrow yRx$

Suppos $x-y=3k$

$-(x-y)=-3k$

$y-x=3\cdot -k$

$xRy \wedge yRz \Rightarrow xRz$

$x-y=3k \quad y-z=3n$

$(x-y)+(y-z)=x-z.$

~~$3k+3k$~~

~~$k+n=b$~~

$x-z=$ ~~[illegible]~~ $3k+3n$

$x-z=$ ~~$3(2k)$~~ $3(k+n)$

$x-z=3b$

(b) Laurin’s solution.

Figure 34: Two good examples of a proof for Question 3.

Niculin’s and Laurin’s solutions are nice examples of the proof, but like other students too, they struggle to choose two different integers in the symmetry proof. They have it written in their proofs here, but they needed help to understand that they should use different integers in the proof.

Question 4

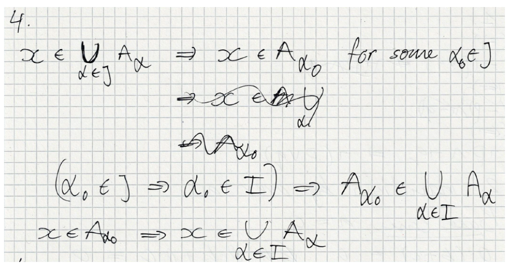
If $J \neq \emptyset$ and $J \subseteq I$, does it follow that $\bigcup_{\alpha \in J} A_\alpha \subseteq \bigcup_{\alpha \in I} A_\alpha$? What about $\bigcap_{\alpha \in J} A_\alpha \subseteq \bigcap_{\alpha \in I} A_\alpha$?

Question 4	Laurin	Nevia	Ladina	Niculin	Lavinia	Nuot	Linard	Nicola	Liun	**Averarge LEAN**	**Mean Non-LEAN**	**Mean**
D	1	1	2	2	2	1	2	2	0	1.4	1.5	1.4
MS	0	1	2	2	2	1	2	2	1	1.4	1.5	1.4
L	1	2	2	1	2	1	2	1	1	1.6	1.25	1.4
HLI	1	2	2	2	2	1	1	1	0	1.2	1.5	1.3
S	0	1	2	1	2	1	1	2	0	1	1.25	1.1
UE	0	0	1	1	2	0	1	2	1	1	0.75	0.9
Total	3	7	11	9	12	5	9	10	3	7.6	7.75	7.7

Table 19: Results from Question 4.

This is the only question in which Non-LEAN students perform (slightly) better than LEAN students. As we stated in Section 3.2, we expect that Non-LEAN students have an advantage over LEAN students in this question.

Below we see Lavinia's approach to the proof. It is the best-scored question in all the interviews, as she uses a very nice example to disprove the statement about the intersection.



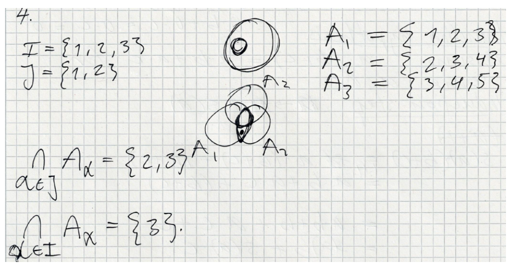
4.

$x \in \bigcup_{\alpha \in J} A_\alpha \Rightarrow x \in A_{\alpha_0}$ for some $\alpha_0 \in J$

$(\alpha_0 \in J \Rightarrow \alpha_0 \in I) \Rightarrow A_{\alpha_0} \in \bigcup_{\alpha \in I} A_\alpha$

$x \in A_{\alpha_0} \Rightarrow x \in \bigcup_{\alpha \in I} A_\alpha$

(a) Proof for the union.



4.

$I = \{1,2,3\}$
$J = \{1,2\}$

$A_1 = \{1,2,3\}$
$A_2 = \{2,3,4\}$
$A_3 = \{3,4,5\}$

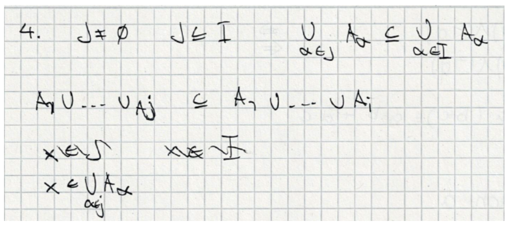
$\bigcap_{\alpha \in J} A_\alpha = \{2,3\}$

$\bigcap_{\alpha \in I} A_\alpha = \{3\}$.

(b) Proof for the intersection.

Figure 35: Lavinia's proof to Question 4.

Lavinia confidently proves the statement about the union and has no problem defining where the chosen x should lie. For other students, both LEAN and Non-LEAN, this could lead to a problem, as one can see in the following figures.

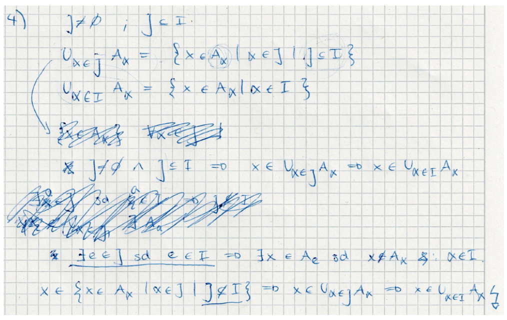

4. $J \neq \emptyset$ $J \subseteq I$ $\bigcup_{\alpha \in J} A_\alpha \subseteq \bigcup_{\alpha \in I} A_\alpha$

$A_1 \cup \dots \cup A_j \subseteq A_1 \cup \dots \cup A_i$

~~$x \in J$~~ ~~$x \in I$~~

$x \in \bigcup_{\alpha \in j} A_\alpha$

(a) Liun's solution.


4) $J \neq \emptyset$; $J \subseteq I$

$\bigcup_{\alpha \in J} A_\alpha = \{x \in A_\alpha \mid \alpha \in J \mid J \subseteq I\}$

$\bigcup_{\alpha \in I} A_\alpha = \{x \in A_\alpha \mid \alpha \in I\}$

[illegible]

$J \neq \emptyset \wedge J \subseteq I \Rightarrow x \in \bigcup_{\alpha \in J} A_\alpha \Rightarrow x \in \bigcup_{\alpha \in I} A_\alpha$

[illegible]

$\exists e \in J$ sd $e \in I \Rightarrow \exists x \in A_e$ sd $x \notin A_\alpha$ $\forall$: $\alpha \in I$.

$x \in \{x \in A_\alpha \mid \alpha \in J \mid J \not\subseteq I\} \Rightarrow x \in \bigcup_{\alpha \in J} A_\alpha \Rightarrow x \in \bigcup_{\alpha \in I} A_\alpha$ ↯

(b) Nuot's solution.

Figure 36: Both students struggle with finishing the proof.

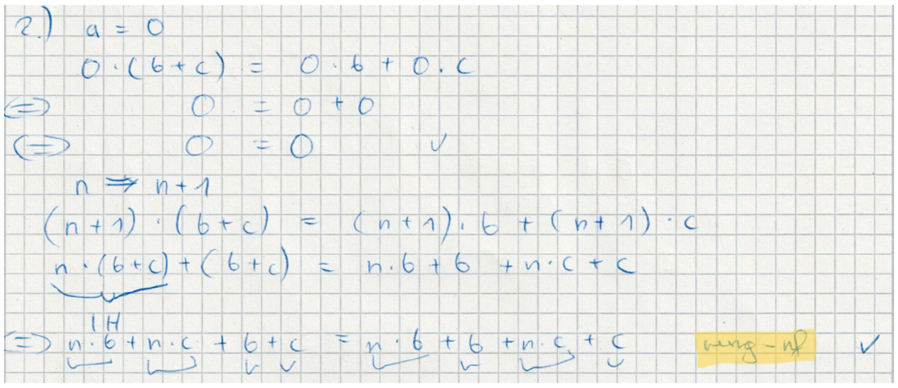
The main reason Nuot is confused, is because he thinks he remembers the first statement from the exercise sheets to be true, a common mistake that first-year students make, namely trying to remember the correctness of statements by heart. While he chooses a certain index e once, to explain that x would lie in some A_α, he then still struggles to define where x lies exactly in the next line. Liun on the other hand cannot see the importance of choosing a specific label α_0 to argue that x would also lie in the union over the index set I. This is why he cannot finish the proof.

This problem with using the variable name given in the statement and not defining new labels is also seen in Question 3, where students get confused in the symmetry proof when they have to choose a different k. While we expect LEAN students to not make these kinds of mistakes, more time working with LEAN is needed to fully get rid of such mistakes. We can see in the questionnaire that the more LEAN students used LEAN at home, the less they would make mistakes like the ones mentioned right now.

Students' General Performance

Having a small data set allows us to now compare the students' performances more carefully. Sometimes we elaborate the students' performance individually, and sometimes we compare LEAN and Non-LEAN students who score similarly in the interview to point out some differences or similarities.

We start discussing Ladina, as she is quite a special case. Ladina has already a lot of experience with LEAN. She has a Bachelor's in informatics and even implemented some formalisations of mathematics in LEAN herself at some time. However, she started studying mathematics as a first-year student just like the others. Coming from this computer science background, one can perfectly see on various occasions that she sees mathematics differently than the other students (or even us).



2.) $a = 0$

$0 \cdot (b + c) = 0 \cdot b + 0 \cdot c$

$\Leftrightarrow \quad 0 = 0 + 0$

$\Leftrightarrow \quad 0 = 0 \quad ✓$

$n \Rightarrow n + 1$

$(n+1) \cdot (b + c) = (n+1) \cdot b + (n+1) \cdot c$

$\underbrace{n \cdot (b+c)}_{IH} + (b + c) = n \cdot b + b + n \cdot c + c$

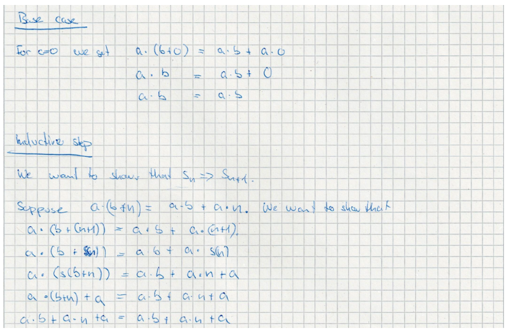
$\Leftrightarrow n \cdot b + n \cdot c + b + c = n \cdot b + b + n \cdot c + c$ ring-nf ✓

Figure 37: Question 2 from Ladina's interview.

At the end of Question 2, Ladina says that here she would just use *ring_nf*, one of the LEAN tactics we discussed before. This is not the only time she mentions a LEAN tactic she would use to solve the next step of a proof.

Even though he is a Non-LEAN student, Nicola scores better than most students in the interview. We do not know much about him, as he did not come to the meetings, but we find out at the beginning of the interview that he is a first-semester student. There are always students who are really good at mathematics, and it should be clear that the difference between LEAN and Non-LEAN students will never be that all LEAN students perform better than all Non-LEAN students. Nicola has a clean and structured proving style without ever having used LEAN. However, he solves the first exercise in the interview with sets, i.e. using set theory, as can be seen in Figure 29. While he solves it correctly, it would have been much easier doing it with propositional logic, which he did not learn in the course "Foundations of Mathematics". In his example, we can see that no matter how well we are at something, learning something new can never be bad.

Laurin and Nuot are the LEAN and Non-LEAN students respectively that score the lowest mark during the interviews. As we understand from the feedback from Laurin, this score may depend on his oral examination phobia. One can also see that in the questionnaire, he is the one who answers with "never" when asked if LEAN was used outside of the meeting, which is why we hesitate to say that LEAN did not improve the mathematical skills in his case. Both Nuot and Laurin have no academic background at the time of the interview.


Base case

For $c=0$ we set $a\cdot(b+0) = a\cdot b + a\cdot 0$

$a\cdot b = a\cdot b + 0$

$a\cdot b = a\cdot b$

Inductive step

We want to show that $S_n \Rightarrow S_{n+1}$.

Suppose $a\cdot(b+n) = a\cdot b + a\cdot n$. We want to show that

$a\cdot(b+(n+1)) = a\cdot b + a\cdot(n+1)$.

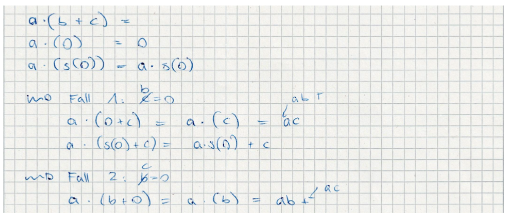
$a\cdot(b+s(n)) = a\cdot b + a\cdot s(n)$

$a\cdot(s(b+n)) = a\cdot b + a\cdot n + a$

$a\cdot(b+n) + a = a\cdot b + a\cdot n + a$

$a\cdot b + a\cdot n + a = a\cdot b + a\cdot n + a$

(a) Laurin's solution.


$a\cdot(b+c) =$

$a\cdot(0) = 0$

$a\cdot(s(0)) = a\cdot s(0)$

$\leadsto$ Fall 1: $\overset{b}{\cancel{c}}=0$

$a\cdot(0+c) = a\cdot(c) = ac$ ab +

$a\cdot(s(0)+c) = a\cdot s(0) + c$

$\leadsto$ Fall 2: $\overset{c}{\cancel{b}}=0$

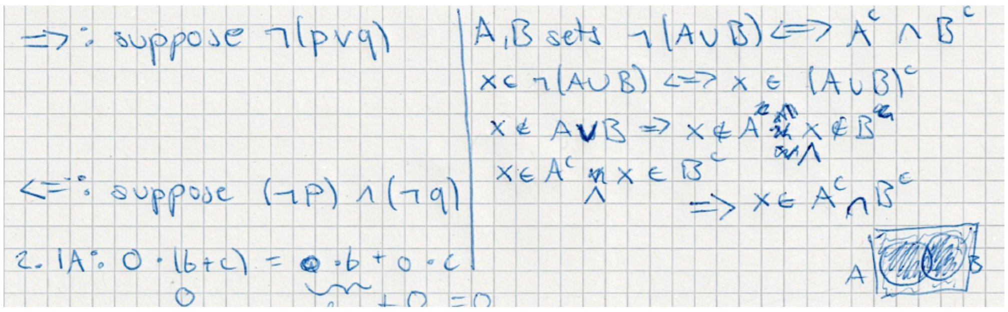
$a\cdot(b+0) = a\cdot(b) = ab +$ ac

(b) Nuot's solution.

Figure 38: A LEAN and Non-LEAN solution to Question 2.

Even though they both score a lower result during the interview than other students, one can see the more structured way Laurin writes his proof in Question 2. This is possibly due to the LEAN level treating natural induction.

Linard is one of the LEAN students with a very high score and has the second-best *Definitions* rating after Ladina. Interestingly, he also tries to solve Exercise 1 from the interview using set theory, as one can see in Figure 39 below. So it seems that first-order logic is not very present in LEAN students too. We would not say that this is due to LEAN not being a competent theorem prover, but rather to the little time we have learning about first-order logic in LEAN.


$\Rightarrow$: suppose $\neg(p \vee q)$

$\Leftarrow$: suppose $(\neg p) \wedge (\neg q)$

2. IA: $0\cdot(b+c) = 0\cdot b + 0\cdot c$

$0 \qquad + 0 = 0$

A, B sets $\neg(A\cup B) \Longleftrightarrow A^c \wedge B^c$

$x \in \neg(A\cup B) \Longleftrightarrow x \in (A\cup B)^c$

$x \notin A\cup B \Rightarrow x \notin A \wedge x \notin B$

$x \in A^c \wedge x \in B^c$

$\Rightarrow x \in A^c \cap B^c$

Figure 39: Question 1 from Linard's interview.

Linard starts Question 1 with a propositional logic approach but later finishes it using sets. Note her clean proof structure and the example she uses at the bottom right. It is one of the very few examples used over all interviews.

Next we would like to discuss Nevia, a Non-LEAN student. Her *Logical thinking* is remarkable, but she struggles a bit with *Definitions* of mathematical objects, like the union of indexed sets (compare to Table 14). Having a bit of practice doing LEAN exercises, she could profit and get a better understanding of those definitions for sure.

Niculin is another Non-LEAN student who scores almost the same mark as Nevia, but he seems a lot more confident. He spends more time in Exercises 1 and 3 than the other students because he gets a bit confused there. But he is pretty secure with definitions and proof structure over the whole interview in general.

A very enthusiastic LEAN student is Lavinia. While she is not as experienced as Ladina, she did most LEAN at home compared to the other students. Her high score in the interview shows the fruits of her commitment. Keep in mind that she is the only student scoring the total 12 points in a question. Even though she never studied another course of study before, she is a computer affine out of her own interest. One can really see some LEAN-thinking during the interview, but unfortunately, she struggles with Exercise 1 too. We would like to mention the way she solves Exercise 4. She uses an excellent example to disprove Subquestion 4. b) and manages to solve this exercise in a more elegant way than others. It is not possible to say if that is thanks to LEAN or not. Except for Ladina, Lavinia is most likely to continue using LEAN regularly during her studies.

That leaves us with Liun, the last LEAN student. He is a very committed student and learner, and he is the one who "praised" LEAN the most during the questionnaire, mentioning that he uses our LEAN exercises often to understand a proof he cannot figure out by himself. With a bit more experience, he can get as good as Lavinia and Ladina. We are not sure if LEAN is more of a helping tool for him or if his interest in proof assistants is also awakened by the sessions, but we hope that he will continue to use LEAN either way.

If we compare the interviews in total, we can see that LEAN students perform slightly better, with outliers in both directions. Our sample is too small to derive a relevant effect of teaching with LEAN, but it is still nice to see that LEAN students perform well in general. As already mentioned before, learning something new that is meaningful is never wrong. But LEAN does not make a mathematical superhero out of everybody, and some mathematicians are incredible without ever having learned about theorem provers.

Significance

In this chapter, we present the tests we do to check whether our results are significant. Our null hypothesis is that the mean performances of LEAN and Non-LEAN students are the same. We calculate the p-value to check if we reject the null hypothesis. As a significance value, we choose $\alpha = 0.05$.

For the interview results and the final exam mark, we do a t-test for two independent samples, LEAN and Non-LEAN students [DAT24c]. We assume the variances to be heterogeneous, which influences the degrees of freedom.

The t-value is computed as

$$t = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}},$$

where $\overline{x}_k$ is the respective mean value for LEAN or Non-LEAN students, s_k^2 is the standard deviation and n_k is the number of students per sample.

The standard deviation is computed as follows,

$$s_k^2 = \frac{1}{n_k - 1} \sum_{i=1}^{n_k} (x_i - \overline{x}_k)^2.$$

Here, x_i stands for the performance of each individual student.

To compute the p-value, we need the number of degrees of freedom. In our case, this number is computed as

$$df = \frac{(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2})^2}{\frac{1}{n_1-1}(\frac{s_1^2}{n_1})^2 + \frac{1}{n_2-1}(\frac{s_2^2}{n_2})^2}.$$

In Tables 20 and 21 below, we present the given data.

$\overline{x}_1$	$\overline{x}_2$	n_1	n_2	s_1^2	s_2^2	t	df
28.8	24	5	4	82.7	104.67	0.73	6.15

Table 20: Quantities from the interviews to find p-value.

$\overline{x}_1$	$\overline{x}_2$	n_1	n_2	s_1^2	s_2^2	t	df
46	34.92	5	52	32.5	181.17	3.51	9.23

Table 21: Quantities from the exams to find p-value.

Using an online tool, we calculate the p-values. For this, we just have to type in the t-values and the degrees of freedom for both tests [DAT24b].

The following p-values were found.

	Interviews	**Exams**
p-values	0.49	0.006

Table 22: p-values of interview and exam performances.

We now see that for the interviews, while the LEAN students do perform better on average than the Non-LEAN students, the difference of the means is not significant. For the exam, however, the p-value is smaller than the significance level. Therefore, here we can reject the null hypothesis and state that LEAN students performed significantly better on average in the exam than Non-LEAN students.

For the t-test, we assume the data to be normally distributed. To consider the case where the data might not be normally distributed, we conduct the nonparametric Mann-Whitney U-test for independent data [DAT24a]. This test needs a lot of intermediate steps. These steps are explained next, and described for both data, interviews and exam results, at the same time.

First, we rank all scores from 1 to n, where 1 is for the lowest score and n for the highest. If there are some tied ranks, we consider the mean of the ranks. For example, if the ranks 6, 7 and 8 all scored the same, they are all assigned the rank 7. We then compute the sum of the LEAN and Non-LEAN ranks respectively, denoted as T_1 and T_2, and we compute a number needed later, which we will call tR (tied ranks)

$$tR = \sum_{i=1}^{k} \frac{t_i^3 - t_i}{12}.$$

Here, k stands for the number of tied ranks and t_i is the number of people sharing the same rank.

T_1	T_2	n_1	n_2	tR
27.5	17.5	5	4	0.5

Table 23: Rank sums and tied rank number from the interviews.

T_1	T_2	n_1	n_2	tR
217.5	1435.5	5	52	26.5

Table 24: Rank sums and tied rank number from the exams.

Next, we calculate the U-values for LEAN and Non-LEAN students and then we choose the smaller of the two to be our U-value. We also compute μ_U and σ_U

$$U_j = n_1 \cdot n_2 + \frac{n_j \cdot (n_j + 1)}{2} - T_j,$$

$$\mu_U = \frac{n_1 \cdot n_2}{2},$$

$$\sigma_U = \sqrt{\frac{n_1 \cdot n_2}{n \cdot (n-1)}} \cdot \sqrt{\frac{n^3 - n}{12} - tR}.$$

In these formulas, j can take the values 1 and 2 for LEAN and Non-LEAN students respectively and n is the sum of both sample sizes.

U	μ_U	σ_U	z
7.5	10	4.07	-0.61

Table 25: Quantities from the interviews to find p-value.

U	μ_U	σ_U	z
57.5	130	35.42	-2.05

Table 26: Quantities from the exams to find p-value.

The z-values in Tables 25 and 26 are calculated using $z = \frac{U - \mu_U}{\sigma_U}$. Once we have that, we use an online calculator to find the p-value [Sta24].

	Interviews	**Exams**
p-values	0.54	0.04

Table 27: p-values of interview and exam performances.

The Mann-Whitney U-test returns the same conclusions as the t-test. The better performance of LEAN students during the interviews is not relevant ($p > \alpha$) but for the exams, our results are significant ($p < \alpha$).

Results from the Questionnaire

Here we present the results of the questionnaire from Chapter 3.2.

Did LEAN motivate you to spend more time on a proof than usual? i.e. would you spend more time trying to solve an exercise in LEAN (or the NNG) than on paper?

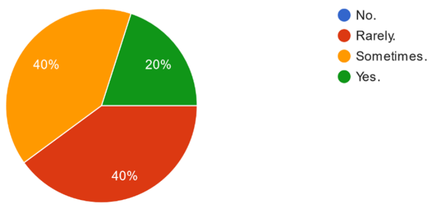


Figure 40: Question 1.

Comments:

- LEAN kind of forces your hand in this case. However, occasionally it also happens by choice, as the exploration of the structure of the proof is more in-depth with LEAN. This is not always the case, as some proofs pretty much can be auto-generated and it also depends on my interest in the proof.
- Especially the NNG.

Did LEAN improve your proving skills in general?

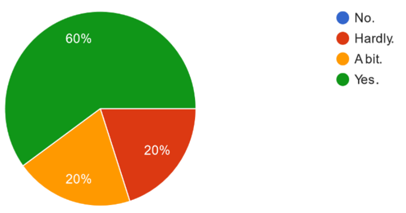


Figure 41: Question 2.

Comments:

- Absolutely, in typical proof patterns (inj/surj proofs, set equality, induction, contradiction, etc.) I've learned to start out in a very mechanical way, almost like mathematical muscle memory. I attribute this to LEAN.
- Helped with structure of proofs.

Did LEAN have an influence on how well you felt prepared for the final exam?

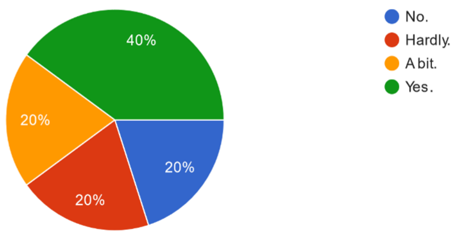


Figure 42: Question 3.

Comments:

- I assumed that I would be able to do the aforementioned mechanical proofs quite a bit faster than if I had not looked at LEAN. However, in general, this was not the case and it showed in the exam. My time in the exam was mostly taken up with annoying algebraic manipulations and manipulations over summation symbols (stuff I use simp for). I blame this on the exam though, as the hard part should not be those aspects.

How often did you use LEAN besides the meetings?

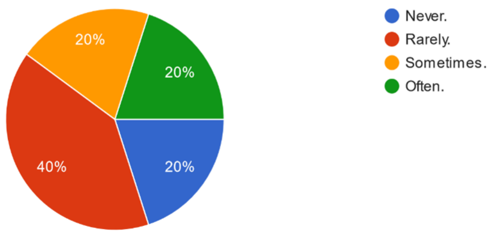


Figure 43: Question 4.

Comments:

- I try to formalize various theorems and am actively doing formal verification of code. Sometimes, when a theorem from one of my modules is not clear, I browse mathlib for it to see all the parameters it takes and the output, so I get a very structured overview of what the theorem exactly does (and I get to see applications of it in mathlib).
- Whenever I was stuck with an exercise I tried to compare with the code given by you.

Should students be taught mathematics with LEAN in the future?

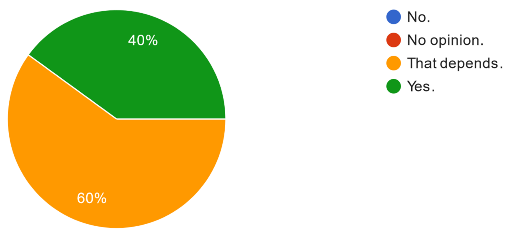


Figure 44: Question 5.

Comments:

- Some people might also find it interesting but it is not integral to the courses in my opinion.
- Yeah because if every student would have a module for lean at university I think there would be more motivation to consult lean (as a help) for a proof instead of using a website like stackexchange an coping a proof.
- Unfortunately, LEAN is a complicated programming language (compared to Python or other imperative languages). While the basics of the tactic language can be taught in a short amount of time, when students need to push outside of those bounds or understand parts of the language itself (idea of lambda calculus, etc.), good knowledge of functional programming and a bit of type theory is required. These things are necessary when formalizing harder theorems, most of the time. So if mathematics is taught with LEAN, it should be taught in conjunction with a course that teaches the language itself. Additionally, some things in mathlib might be too complicated for people just starting out with mathematics (for example that there is no extensive notion of a vector space there, everything is formulated in terms of semi-modules).
- I think it should be an option after the gymnasium level, and maybe it could even be a nice optional subject at the (mathematical) gymnasium. But I do not think that it should be mandatory.
- Yes, I think it would encourage students to work on their proofs more than currently (since you don't need to be able to do proofs to actually pass analysis/linear algebra).

Open question: Do you plan to continue working with LEAN during your studies? Why or why not?

Comments:

- I hope so when I have more time.
- Probably not often but I can see myself using it in some instances where I think that a lean proof would help my understanding of a certain mathematical proof.
- Yes, I enjoy it a lot and it generally benefits my understanding of ideas and concepts. It's a lot of fun and it creates even more 'click' moments than just pen and paper (and hence a dopamine rush :D)
- No I do not think so, as I prefer not working with the computer and I want to learn and be able to do the proofs by 'hand'.
- Yes, I will try to. For example for things based on set theory this seems very effective. It's easily understood and you get feedback whenever you're stuck.

Open question: Would you have done something differently in the way I held my meetings with you?

Comments:

- For exercises that needed some specific theorem to solve it would sometimes have been useful to have a hint like 'use [theorem_something] in the proof' as a comment because not knowing those was often the part that got me stuck and had me just looking at the solutions.
- No I think the meetings where always something to look forward also regarding the very relaxed and open atmosphere during them. The structure was also very good and always matched with what was happening in "Foundations of Mathematics".
- It's hard to really find fundamental improvements for the meetings with the little amount of time we had. I'm not sure how it was with the other students, but for me it was a nice experience, so I do not really have anything to improve.
- Maybe a touch more theory, would have helped me for example.
- I don't think so, but looking back I think if the lectures started off with constructing the natural numbers (such that we could've started with the NNG) it might've been easier to get into it.

5 Discussion

We discuss some general results from the previous section here. This chapter also contains the experience I had both with learning and teaching (with) LEAN. For each of the two, we also write some pros and cons. For the latter part, we mostly use the first-person singular perspective and write in past tense.

5.1 Interviews

Here we talk about general observations which are not that relevant for the results section.

When reading the interview discussion, one can see that some students are more nervous than others. Apart from Laurin, who mentions his oral examination phobia, the Non-LEAN students are in general a bit more excited than the LEAN students. While one can argue that learning LEAN makes one a more confident mathematician, we are very careful here with making that assumption. The LEAN students know the interviewer from the meetings they had in the past semester. So Non-LEAN students may be more nervous because they do not know the interviewer as well as the LEAN students.

Some of the criteria we observe in the interview are not generally performed better by LEAN students, e.g. the use of *Mathematical Symbols*, *Logic* or *Use of Examples*. Since LEAN requires these criteria to be done well, we assume that LEAN students still profit from proving statements with LEAN, but that they need to spend more time doing it to understand the importance of also working so precisely and thoroughly on paper.

The *Use of Examples* criterion is performed weakly over almost all interviews. This is nothing uncommon. As stated in [TI21], the *Use of Examples* is in general a criterion of a good proof that neither LEAN nor Non-LEAN students use often. In some cases, it does not even make much sense to use examples. But it would still be a good thing to encourage students to use examples when they struggle with a proof in the future.

We would also like to talk about students who came to one or two meetings and then stopped. The feedback from them is always identical. They all say, that they do not have enough time at the moment to learn LEAN, and although they are interested in learning it, they cannot do it at the time, also due to the low relevance to their studies. We discuss this issue in more detail in Section 5.2. If we cannot integrate learning theorem provers into the curriculum, we must at least make students understand the benefits of studying them. We would never criticize students for not coming to the meetings, as we know very well how stressful the first semester can be, but we would recommend them all, to pick up learning a theorem prover at some point during their studies.

5.2 My Experience

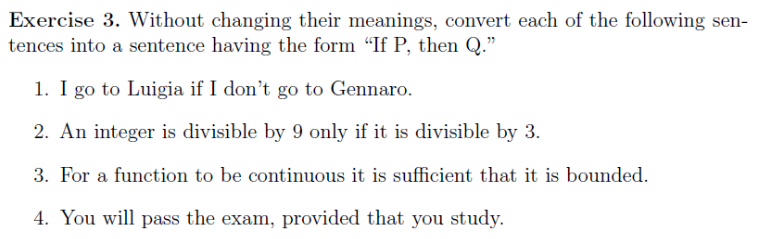
We start this section by discussing the experience I had learning LEAN in the process of this thesis. After arguing about the pros and cons of using LEAN as a theorem prover, we end this section talking about my experience teaching (with) LEAN and reason why others should do that too.

Implementation - Learning LEAN

One of the main parts of my work was implementing the questions of the weekly exercise sheets from the "Foundations of Mathematics" course into LEAN. There were seven exercise sheets in total with 4 - 6 questions each. I would always get an exercise sheet simultaneously with the students, which means that I always had two weeks to finish implementing the whole sheet before the next came out. Since the course topic changed a lot, as one can see in Chapter 2.1, implementing the questions was quite challenging because I had to learn new notations in LEAN each week. For example, between the 4th and the 6th week of the sessions, I had to learn how to implement an induction proof, a strong induction proof, proofs for relations and proofs for functions.

I started learning LEAN in February of 2023. At that time, I watched YouTube lectures about Haskell and logic in Haskell [Fur23]. I was also reading the scripts of a LEAN seminar that had been held at UZH a year before the beginning of my thesis [ST22]. It was difficult to understand anything at that time. The best introduction to LEAN was playing the Natural Number Game. I became addicted to solving the levels, and although it was a bit difficult in the beginning, I learned fast. A month later, I had mostly finished the NNG and I had tried to learn a bit about first-order logic, although it did not make much sense at that time. I watched a lot of Kevin Buzzard's talks about how to teach with LEAN. From the 18th to the 20th of April 2023, I participated in an online seminar with people who had experience teaching (with) LEAN [AV23c]. I do not want to explain in detail what was discussed during the seminar, I just summarize the most significant experiences I had. Interested readers can watch the talk here [Sch23]. A lot of the presenters talked about one problem teaching with LEAN. Namely, that the theorem prover can be distracting and that first-year students are overwhelmed when they have to learn new mathematics and LEAN. This is why Patrick Massot, one of the presenters and an eminent authority in the LEAN community, prepared his course without type theory, meaning that he developed his syntax based on LEAN so that the students can do mathematics with LEAN while keeping their known syntax. While this is certainly a good idea, it requires a lot of knowledge and work to pull off. Another thing that was often mentioned was the use of digital classrooms to hold the LEAN lessons. I talk about such possibilities and everything from this seminar concerning my meetings in Chapters 5.2 and 5.2. Some also explained how they do the grades of their courses. Since my meetings were completely voluntary, this did not concern me. However, I think that with an obligatory course, one would certainly encourage active work with LEAN. Dr. Paola Iannone and Dr. Athina Thoma mentioned that two students in their study using features of LEAN would engage very differently, which is an observation I appreciate. I do not wish for students to become machine-like thinkers who all use LEAN tactics in the same way. Individuality is key for the future of mathematics. One last thing I want to mention is students' input during the seminar saying without voting against, that the NNG was a perfect introduction to LEAN, which is something I thought so too. While I could not start with the NNG with my students, since the natural numbers was one of the later chapters of their course, unfortunately, I used it when we introduced the natural numbers on our own and had great experiences with it. Over the summer I worked through formalising mathematics in LEAN [Buz22]. It is a LEAN script that helps one get familiar with some notations in LEAN. Unfortunately, it is written in LEAN 3. All the scripts, YouTube tutorials and talks I had processed by summer were for LEAN 3, as LEAN 4 was still developing. This led to a problem when I started teaching (with) LEAN, as can be seen in Chapter 5.2. I also needed to learn how to install LEAN on my device, and I wrote an instruction to do so. To make the material accessible for my students, I had to teach myself basic GitHub since I never had used it before. Finally, I was ready to teach some LEAN myself. On the 2nd October of 2023, I had my first meeting.

The first two exercise sheets (Sheet 0 and Sheet 1) had a lot of questions where one could just do logical thinking which did not make any sense to implement into LEAN. For an example, see the Figure 45 below.



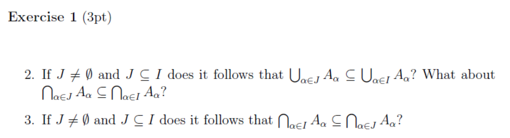
Exercise 3. Without changing their meanings, convert each of the following sentences into a sentence having the form "If P, then Q."

1. I go to Luigia if I don't go to Gennaro.
2. An integer is divisible by 9 only if it is divisible by 3.
3. For a function to be continuous it is sufficient that it is bounded.
4. You will pass the exam, provided that you study.

Figure 45: Question 3 from Exercise sheet 0 [AW23a].

Therefore there were only two questions in total to implement for these exercise sheets, which is why I added some introduction examples of my own to get used to propositional logic in LEAN and both deMorgan laws since they fit well into this environment. It was a good thing that the first two exercise sheets did not have so many exercises, as it gave me the possibility to introduce the students to the surroundings of LEAN.

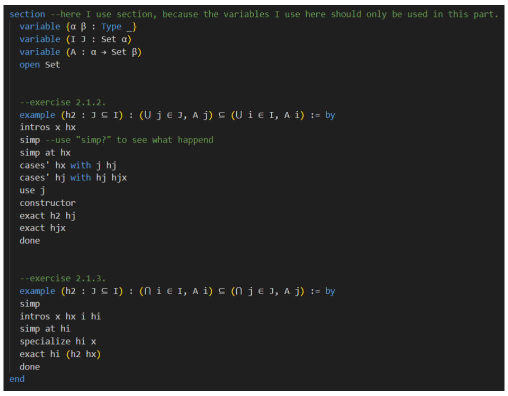
Exercise sheet 2 already used some advanced theorems and surroundings of LEAN's Mathlib [AV23d]. Especially Question 1 in Figure 46 below.



Exercise 1 (3pt)

2. If $J \neq \emptyset$ and $J \subseteq I$ does it follows that $\bigcup_{\alpha \in J} A_\alpha \subseteq \bigcup_{\alpha \in I} A_\alpha$? What about $\bigcap_{\alpha \in J} A_\alpha \subseteq \bigcap_{\alpha \in I} A_\alpha$?
3. If $J \neq \emptyset$ and $J \subseteq I$ does it follows that $\bigcap_{\alpha \in I} A_\alpha \subseteq \bigcap_{\alpha \in J} A_\alpha$?

(a) Question 1 of Exercise sheet 2 [AW23b].

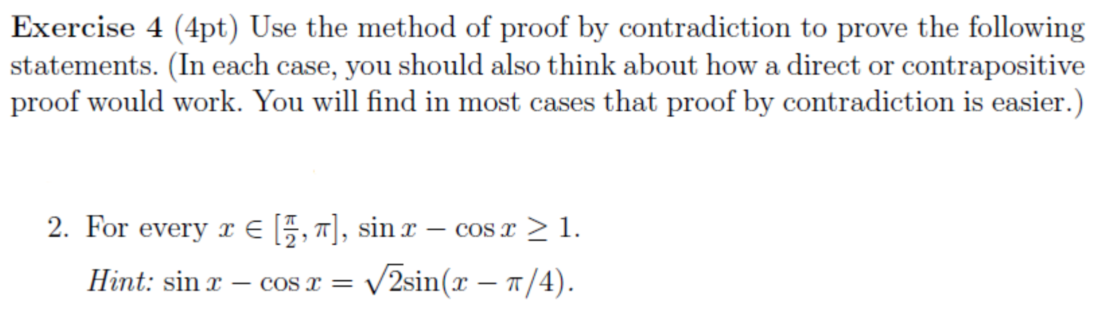


```
section --here I use section, because the variables I use here should only be used in this part.
  variable {α β : Type _}
  variable (I J : Set α)
  variable (A : α → Set β)
  open Set


  --exercise 2.1.2.
  example (h2 : J ⊆ I) : (⋃ j ∈ J, A j) ⊆ (⋃ i ∈ I, A i) := by
  intros x hx
  simp --use "simp?" to see what happend
  simp at hx
  cases' hx with j hj
  cases' hj with hj hjx
  use j
  constructor
  exact h2 hj
  exact hjx
  done


  --exercise 2.1.3.
  example (h2 : J ⊆ I) : (⋂ i ∈ I, A i) ⊆ (⋂ j ∈ J, A j) := by
  simp
  intros x hx i hi
  simp at hi
  specialize hi x
  exact hi (h2 hx)
  done
end
```

(b) The proof and structure in LEAN.

Figure 46: The structure of unions and intersections in LEAN.

I was not able to finish the whole proof in LEAN in the two weeks I had, but it turned out that having only a 45-minute meeting each week we could not cover all the exercises anyway. That was the moment I decided to implement all possible questions but cover only the simpler ones during the meetings. The students would get the LEAN implementations with solutions as fast as possible to try and understand them on their own. There were also exercises in this sheet that did not make sense to implement into LEAN at all e.g. in Figure 47 below.



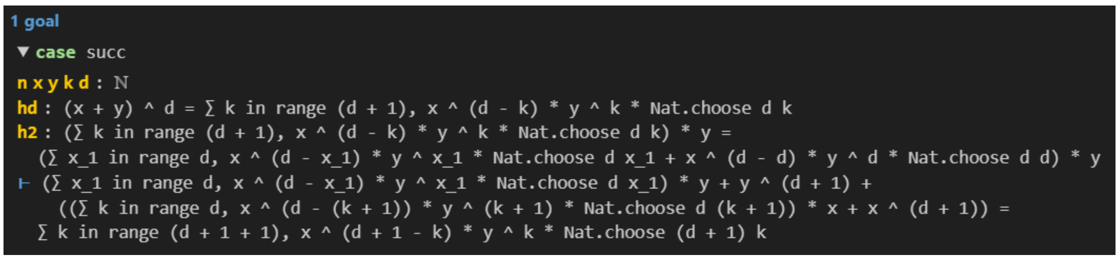
Exercise 4 (4pt) Use the method of proof by contradiction to prove the following statements. (In each case, you should also think about how a direct or contrapositive proof would work. You will find in most cases that proof by contradiction is easier.)

2. For every $x \in [\frac{\pi}{2}, \pi]$, $\sin x - \cos x \geq 1$.
   Hint: $\sin x - \cos x = \sqrt{2}\sin(x - \pi/4)$.

Figure 47: Question 4.2 from Exercise sheet 2 [AW23b].

Teaching myself and the students how to work with trigonometric functions and the number π in LEAN would have taken two sessions on their own.

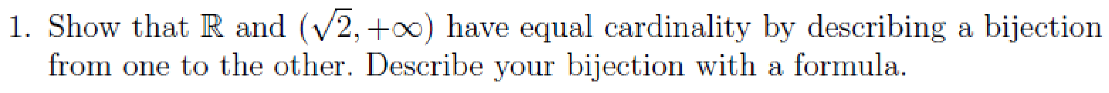
Exercise sheet 3 was about mathematical induction. This topic works very well in LEAN and helps the students understand the structure of a mathematical induction proof clearly. However, the expressions that needed to be proved by induction in the exercise sheet, were so complicated to deal with in LEAN, that I decided to add some simple induction proofs at the beginning of my level. One exercise of this sheet was to prove the binomial theorem. The expressions in this proof became so long, that it was very difficult to even see the goal of the proof, as one can see here.



```
1 goal
▼ case succ
n x y k d : N
hd : (x + y) ^ d = ∑ k in range (d + 1), x ^ (d - k) * y ^ k * Nat.choose d k
h2 : (∑ k in range (d + 1), x ^ (d - k) * y ^ k * Nat.choose d k) * y =
  (∑ x_1 in range d, x ^ (d - x_1) * y ^ x_1 * Nat.choose d x_1 + x ^ (d - d) * y ^ d * Nat.choose d d) * y
⊢ (∑ x_1 in range d, x ^ (d - x_1) * y ^ x_1 * Nat.choose d x_1) * y + y ^ (d + 1) +
    ((∑ k in range d, x ^ (d - (k + 1)) * y ^ (k + 1) * Nat.choose d (k + 1)) * x + x ^ (d + 1)) =
  ∑ k in range (d + 1 + 1), x ^ (d + 1 - k) * y ^ k * Nat.choose (d + 1) k
```

Figure 48: Question 3 from Exercise sheet 3 in LEAN.

Relations were covered at the end of Exercise sheet 3 and the beginning of Exercise sheet 4. Once I figured out how to implement relations, the proofs were quite easy to handle in LEAN. But right after relations came functions. To prove statements about functions, the teaching assistant of the course chose explicit examples, as seen in Figure 49 below.



1. Show that $\mathbb{R}$ and $(\sqrt{2}, +\infty)$ have equal cardinality by describing a bijection from one to the other. Describe your bijection with a formula.

Figure 49: Question 1 from Exercise sheet 4 [AW23c].

Implementing the proof of explicit examples turned out to be a real challenge in LEAN. I spent hours searching for the right theorems and notations to implement the proof like I intended to do and it was only with the help of Ladina that I managed to prove all of the statements I implemented.

Even harder than the questions of Exercise sheet 4 were some of the questions of Exercise sheet 5 about the cardinality of sets. That is why I decided to skip one meeting and implement a repetition level about relations, functions and cardinality of sets with some simpler exercises that I chose. In the Chapter 5.2, I talk about this decision more in detail. There was a ray of light though. During this exercise sheet, the students started with the natural numbers, which are straightforward to prove in LEAN. I could teach the students the natural numbers using the Natural Number Game [BE23]. So half of the exercise sheet was a nightmare to implement in LEAN and half was a piece of cake.

Finally, Exercise sheet 6 was simpler again [AW23d]. The first question was about the multiplication of natural numbers, which was straightforward to implement. Then there were three questions about order relations and after searching the Mathlib library for the structure of some order relations, it was quite easy to implement the proofs. However, they use a certain structure in LEAN which is not trivial. I spent more time explaining to the students the LEAN structure of the proof than for the proof itself. Below, one can see this structure.

```
def Div_by_n
  (a b : ℕ)
  : Prop := a ∣ b


example : IsPartialOrder ℕ Div_by_n := by
haveI isrefl : IsRefl ℕ Div_by_n := { -- a ∣ a
  refl := by
    intro a
    exact dvd_refl a
    done
}
haveI istrans : IsTrans ℕ Div_by_n := { -- a ∣ b ∧ b ∣ c → a ∣ c
  trans := by
    intros a b c
    intro hab
    intro hbc
    exact dvd_trans hab hbc
    done
}
haveI antisymm : IsAntisymm ℕ Div_by_n := { -- a ∣ b ∧ b ∣ a → a = b
  antisymm := by
    intros a b
    intro hab
    intro hba
    exact dvd_antisymm hab hba
    done
}
haveI ispreorder : IsPreorder ℕ Div_by_n := IsPreorder.mk --put together
exact IsPartialOrder.mk
done
```

The difficulty lies in knowing the several instances that one needs to prove for example that something is a partial order. We need to use the *haveI* tactic, prove all instances individually, and then put everything together to form a partial order.

After the semester, I spent three whole days going through all the questions implemented, finishing their proofs and/or writing comments next to the exercises so that the students had a chance to solve the levels without having to consider the solutions all the time.

In the end, I programmed seven levels in total. One per each exercise sheet (Levels 1 to 6, with Exercises sheets 0 and 1 combined into one) and a repetition level. All these levels are on my GitHub page [Bot23].

Pros and Cons of Learning LEAN

In general, there is nothing wrong with learning something new. But we can ask ourselves, why we should use LEAN instead of other interactive theorem provers like *Agda*, *Coq* or *Isabelle* to learn assisted proof writing. *Agda*, released in 1999, is a theorem prover based on Zhaohui Luo's unified theory of dependent types (UTT), which is similar to Martin-Löf Type Theory and is written in a way similar to Haskell language [Aug+09]. The problem with *Agda* is that it has no proof automation, which means that it does not have tactics that help us solve proofs as well as LEAN does [AV24]. First released in 1986, *Isabelle* is a well-known higher-order logic (HOL) theorem prover [Pau86]. The problem with *Isabelle* is that it does not have dependent types. Without dependent types, one can just do propositional logic, but not predicate logic. One still can build dependent types in *Isabelle*, but this is complicated. This leaves us with *Coq* and LEAN, which both share the same features [AV24]. They have some small interface and definitional differences, but they are very technical and not worth deeper discussion in this thesis. One can read some discussions about whether LEAN or Coq is preferable in [AV19], [Com24], [Com23b] and [AV23a]. One notices, that opinions are quite opposite and no favourable theorem prover can be decided. It is important to mention that LEAN is not necessarily the best proof assistant. Depending on one's background, one might fancy another one. But it is true, that LEAN has found a big hearing amongst mathematicians, also thanks to the LEAN community. I for myself can say that as a mathematician with no computer science background had little to no problems getting into LEAN.

Learning LEAN improves one's structuring of mathematical proofs. This was shown in [TI21] and could also be seen in the interviews, see Chapter 4. Its interaction feature allows us to do the proof ourselves but with being in this strict corset of the way through. So we have a very good mixture of standalone and assisted proving. LEAN also teaches us to work thoroughly and consider **all** cases that need to be checked. Even famous fields-medalist Terence Tao was able to spot a missing case in one of his papers using LEAN [Tao23].

There is one problem though that can occur while learning LEAN. Since we do not want to always prove everything from scratch, some tactics do several things at a time, e.g. the *simp* tactic iteratively goes through all the identities in LEAN's Mathlib with the *@simp* attribute and tries to apply them [Avi+23]. While it is possible to find out what was changed and what hypotheses were used by adding a question mark after a tactic, there is a strong temptation to neglect that and just go on simplifying without properly understanding what is going on. Doing so could mean that we have not understood each step of our proof thoroughly.

Even though that cannot necessarily be considered a con of LEAN, it is rather difficult to learn how to efficiently find adaptable theorems. It takes time to master a coherent procedure to find the right theorems and know what to do as a next step. LEAN helps us with tactics like *exact?*, *apply?* or *rw?* that can show us the next step. However, the proposed theorems do not always correspond to the one we are looking for. Mathlib contains all of LEAN's theorems. Some theorems have a description that helps us figure out what they do. But most of the entries just contain the mathematical definition. For undergraduate students, it might be hard to keep the overview and actually understand what a theorem does and when it can be used. Another difficulty that can occur is when we know what theorem we are looking for but are not sure about what the name should be. I remember searching very long for a theorem to change the third binomial formula $(a-b)(a+b) = a^2 - b^2$. I searched using names like "third_binom" or "binomial_formula". The actual name of that theorem was *rw_sq_sub_sq*. Fortunately, with more experience, it gets easier to find the wished theorems.

Back when there was LEAN 3, Kevin Buzzard, being a lead role model in LEAN, among others, pushed using LEAN for teaching. That way, a lot of teaching material for students was created, simplifying learning LEAN. When I started this thesis LEAN 4 was being developed and at the beginning, I did not see any teaching material using LEAN 4. I thought that since LEAN 4 was in active development, the mathematical community was focused on transferring Mathlib to LEAN 4 and making it even bigger and that interest in providing useful material for learning LEAN 4 would not be developed. In the meantime, however, numerous learning materials have arisen, the LEAN webpage now even has a subpage where people can contribute their teaching material [AV23e]. So LEAN remains a proof assistant with a lot of beginner material, that helps one get into it.

Studies with Students - Teaching (with) LEAN

For my thesis, I taught some students how to prove mathematical statements in LEAN. We decided to work with students from the “Foundations of Mathematics” course at UZH. I prepared a short presentation and introduced myself to the students of the course, telling them my intentions. I then prepared a Doodle and invited all students to participate in weekly sessions, where I would teach them the topics from the “Foundations of Mathematics” course in LEAN. Out of the twelve from 78 students of the course who showed interest in my sessions, seven ever came to a meeting. One of these seven had to drop out of the sessions for personal reasons, unfortunately. I still gave him access to my material, but I do not think that he managed to look at it. Another hardly ever came to my sessions, without stating any particular reasons. All students from the study group were first-semester mathematics students.

In this group of five remaining students, there was one student who already had prior knowledge of LEAN. After the first meeting, I realized that Ladina would be strongly unchallenged next to the others. This is why I proposed to her that we should meet on Mondays and talk about the harder proofs to implement in LEAN. I would meet with the rest of the group on Wednesdays for 45 minutes and teach them the necessary tactics and theorems to solve the exercise sheets from the course and some extra questions with LEAN.

The meetings were all structured more or less the same. I would always start with the goals of today’s meeting, show the students some motivations for why we would be learning today’s topics, recap shortly the theory that we would be needing, show them the questions from the exercise sheets that we would solve and then support them solving the questions in LEAN. For simple questions, I would just give them the question and they would need to try and prove it on their own. The harder questions were given to them with the solutions and we would change between me explaining some steps and them trying to recreate some steps on their own. The sessions were held considering the teaching methods discussed in Chapter 3.1.

The room I could use for my meetings was perfect for teaching with digital media. It had a large table where the students could sit left and right, I was at the top of the table, where there was a big screen I could use to show the students the LEAN code that I had written and the presentation. The students could perfectly see to the front and at each other, which allowed a very learning-efficient environment. I could always walk around the table to watch students progress. With such a small group, it could have been a possibility to work in a shared digital classroom, where we could have seen each other’s screens. My knowledge about this was not good enough to prepare something like this, unfortunately, but we still managed to have a good time. As usual, teaching in a small group is always easier than with a lot of students, since one can be “everywhere at once”. With a bigger group, shared screening would probably be necessary to work efficiently.

During the first weeks, I wrote a document on how to install LEAN 3 and LEAN 4 on one’s device. I was planning to teach the students LEAN 3 since the community was still just migrating to LEAN 4 at that time. Unfortunately, they had already stopped supporting LEAN 3 by the time I had started my sessions [MW23], so almost nobody of my students managed to install it. A huge part of the manual I had written was useless, and I had to change two of my already implemented levels to LEAN 4. After getting used to LEAN 4, rewriting the levels was not so hard, fortunately. But it still took me some time to fix this. Other than this issue with installing the software, I cannot recall any other troubles when teaching with LEAN. My experience with media-based teaching was positive and satisfying.

There were eleven meetings and we met ten times in total, as I once had to just send the students the presentation and the material because I had caught the flu. We started in the third week of the semester and the first few weeks were all as explained before. In week 11, when Exercise sheet 5 came out, I realized that the students were overwhelmed with the large number of new tactics, structures and theorems they had to learn. This can heavily influence their intrinsic motivation in a bad way [EW02]. So I decided to skip a meeting and prepare a repetition level for them instead. The week after, they started to study the natural numbers in the course. For that, I taught using the Natural Number Game, as explained in Chapter 2.2, since all the questions of the exercise sheets about natural numbers were also levels of that game. My repetition level was postponed to the last meeting in the last week of the semester.

Students in my study group were not obligated to join each meeting. The attendance rate was about 80%. Most of the students had an excuse when they could not join a meeting. Especially at the end of the semester, some were either sick, as was I once, or were too busy with regular university stuff. I am happy to say that there was never one week where I taught less than four students in my study group. The atmosphere during the sessions was always great and we never had a single problem or argument during our study time. I am very glad to have found these students who were willing to learn LEAN with me voluntarily and without any compensation.

As an ongoing mathematics teacher, I really enjoyed teaching the group. Even though preparing for the meetings while also learning LEAN myself was stressful sometimes, I would always be happy to hold the lesson I had prepared. I would not change a lot about the sessions in general, except to find a way to use shared screens during the meetings, and the students' feedback was almost only positive, as one can see in Chapter 4. Having more than 45 minutes a week would certainly also increase the quality of the sessions. With that possibility, I would have done more explorative teaching, where the students would have been given chances to learn something on their own. My sessions were a bit too instructive in my opinion and a good mix between instructive and explorative teaching is the key to good education. Even though there were parts where the students could try things on their own, it was still always clear what they should explore exactly, meaning it was quite strict. One must not forget however, that explorative teaching takes time and effort from the students, and with all the other demands university gives their students, I could not have requested this effort from them even if I had wanted to.

In the next step the students of my study group were compared to the rest of the class. I have received permission to compare the achievement in the exercise sheets and the marks of the exam of the whole class. I also did some interviews with students who did not participate in my sessions and students who did. More about the interviews and the results of the meetings were discussed in Chapters 3.2 and 4.

Pros and Cons of Teaching (with) LEAN

Teaching mathematics with LEAN surely has its benefits. I have seen this during my sessions and the following interviews, as we see in Chapter 4 and other sources state that too [TI21]. The structural component encourages clean working while proving and the interactive theorem proving gives a fun twist that motivates us to continue working on the proofs. Students seem to profit from this and generally perform better in proving mathematical statements.

LEAN can make teaching more interesting and diverse, but the problem is that it is neither suitable for current high school or university teaching. For high school mathematics, LEAN is too advanced. Teachers in Switzerland are already complaining about overcrowded curricula and many topics are being removed from class [Wam20]. Trying to fit in abstract mathematics like first-order logic, natural induction or the axiomatic building of natural numbers would just push the boundaries. On the contrary, the university way of teaching is so heavily based on frontal teaching, that it would need a massive restructuring of teaching in universities to teach with LEAN, especially on the undergraduate level. Some brave professors like Kevin Buzzard or Patrick Massot have done that, but they stay a minority [AV23b].

In my opinion the important question is not if LEAN is good for teaching mathematics but rather if LEAN is compatible with our current curriculum. The answer to that is no. During implementation of the questions from the exercise sheets and also in my meetings I could see that students were often overwhelmed with all the theorems used. For example, when we needed to manipulate an equation to prove something, it would take several lines of code just to bring the equation to the desired form, considering that we do not want to use tactics like *ring_nf* or *simp* all the time, as one can see below.

```
/-Use the method of direct proof to prove the following statements.
Let x, y ∈ ℝ. If x^2 + 5y = y^2 + 5x, then x = y or x + y = 5.-/
example (x y : ℝ) :  (x ^ 2 + 5 * y = y ^ 2 + 5 * x) →
((x = y) ∨ (x + y = 5)) := by
intro h
rw [← sub_eq_zero] at h
rw [← sub_eq_zero] at h
rw [← sub_sub] at h
rw [add_comm] at h
rw [← add_sub] at h
rw [add_comm] at h
rw [_root_.sq_sub_sq] at h
rw [sub_zero] at h
rw [sub_eq_add_neg] at h
rw [add_assoc] at h
rw [add_comm (5*y) (-(5*x))] at h
rw [← add_assoc] at h
rw [← sub_eq_add_neg] at h
rw [sub_add] at h
rw [← mul_sub] at h
rw [← sub_mul] at h
rw [ _root_.mul_eq_zero] at h
cases' h with h1 h2
right
rw [sub_eq_zero] at h1
exact h1
left
rw [sub_eq_zero] at h2
exact h2
done
```

While we were able to overcome this using the *have* tacti, as seen in the example at 2.2.49, one still needs tactics like *ring_ nf* to prove the statement we want to have.

There is another problem students (and even I) struggled with. During Exercise sheet 5, the students learn about the Cantor-Bernstein-Schröder theorem which we already mentioned in Theorem 2.1.46). On paper, students would need to find two injective functions and then they could finish their proof with: "Using Cantor-Bernstein-Schröder theorem, we can prove that the sets A and B have equal cardinality.". But finishing the proof in LEAN we need to type this:

```
exact Function.Embedding.schroeder_bernstein f_5_3_is_injective
   f_5_3_inv_is_injective
done
```

which is a lot less intuitive and difficult if one does not know LEAN's name for that theorem by heart. So we do not just have to teach how to do mathematics with LEAN but also how to do LEAN itself. There is no way around that. This on the other hand implies that we would need to change our curriculum at universities in order not to overwhelm students. A whole restructuring of the study would be needed, but maybe with an advantage that needs to be considered.

Given this problem that we would need to also teach LEAN thoroughly, I concluded that maybe we should teach LEAN hybrid. Meaning that some parts would be solved in LEAN, and others would be solved on paper. In the case of the Cantor-Berstein-Schröder theorem above, students would prove with LEAN that the two functions are injective, and then they would write on paper that they now can finish the proof using the Cantor-Berstein-Schröder theorem. That way one could profit from the best of both ways. Using the *sorry* command in LEAN makes that possible.

6 Conclusion and Future Works

In this thesis, we learned how to prove statements using the LEAN theorem prover, taught five volunteer students from the “Foundations of Mathematics” course how to use LEAN, and then compared their proof writing performance with the rest of the class. The group of LEAN students was rather small, so the results are not too relevant. We could still see a slightly better performance from the LEAN students compared to the others. This suggests that teaching LEAN at the beginning of a study of mathematics could indeed lead to a better understanding of proofs.

Teaching and learning LEAN at the same time allowed us to see both sides of the coin. We would not recommend others to do the same in such a short time. It is better to learn LEAN first, and then prepare material to teach it to others. Learning LEAN takes time. All the new notations and the style of proof have to be learned. One can certainly offer better teaching when having the time to think about the best way to lead others to the new syntax. One of the things to keep in mind is that the exercises given to the students should be tailored to be solved with LEAN. This means that students should be taught more general than explicit results. For example, the functions that students have to prove to be bijective should be general functions such as compositions of functions, rather than finding explicit functions like a bijection between $\mathbb{R}$ and $(\sqrt{2}, \infty)$. We think that having students prove general results would even increase their understanding of mathematical structures.

It is our opinion that teaching mathematics with LEAN would be beneficial. However, we argue that in the current state of university teaching, it is just not possible to do so. There needs to be a revision of studies and a modernisation of teaching. We are talking about more flexible and transversal curricula and more digital teaching. We do not mean that students should stay at home and teach themselves on a digital platform, but rather that notebooks should be used more often than just for taking notes in the classroom. This obviously cannot be done in one day, but a good start would be to offer lectures about LEAN or other theorem provers at the university and to integrate these lectures gradually into the main courses.

Theorem provers are the future of mathematics, no question about it. So why not prepare students for that future? Even if it sounds impossible right now, we should consider moving away from set theory, at least after some time, and consider other foundations such as the calculus of constructions. Our ultimate goal should be to teach students homotopy type theory, a type theory that finds correlations between types and homotopies and is currently much discussed between computer scientists and mathematicians. LEAN has even considered using homotopy type theory as its foundation [AV17]. We do not discuss this theory in detail here, but interested readers should definitely check out “Homotopy Type Theory: Univalent Foundations of Mathematics” [Pro13].

References

[Aug+09] Lennart Augustsson et al. *Agda: An Interactive Proof Editor*. 2009. URL: https://web.archive.org/web/20111008115843/http://ocvs.cfv.jp/Agda/index.html (visited on 28/01/2024).

[AV17] A.V. *Theorem Proving in Lean*. 2017. URL: https://leanprover.github.io/tutorial/ (visited on 25/05/2024).

[AV19] A.V. *Address a "setoid hell"*. 2019. URL: https://github.com/coq/coq/issues/10871 (visited on 27/04/2024).

[AV23a] A.V. *(Dis)Advantages of basing a proof assistant on CH correspondence?* 2023. URL: https://proofassistants.stackexchange.com/questions/2449/disadvantages-of-basing-a-proof-assistant-on-ch-correspondence (visited on 27/04/2024).

[AV23b] A.V. *Courses using Lean*. 2023. URL: https://leanprover-community.github.io/teaching/courses.html (visited on 15/01/2024).

[AV23c] A.V. *Learning Mathematics with Lean 2nd event*. 2023. URL: https://sites.google.com/view/learning-maths-with-lean2/presentations-abstracts (visited on 29/04/2024).

[AV23d] A.V. *Mathlib Documentation*. 2023. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib (visited on 15/01/2024).

[AV23e] A.V. *Teaching resources*. 2023. URL: https://leanprover-community.github.io/teaching/resources.html (visited on 15/01/2024).

[AV24] A.V. *Proof assistant*. 2024. URL: https://en.wikipedia.org/wiki/Proof_assistant (visited on 28/01/2024).

[Avi+23] Jeremy Avigad et al. *Theorem Proving in Lean 4*. 2023. URL: https://leanprover.github.io/theorem_proving_in_lean4/title_page.html (visited on 15/01/2024).

[Avi19] Jeremy Avigad. 'Learning logic and proof with an interactive theorem prover'. In: *Proof technology in mathematics research and teaching* (2019), pp. 277–290.

[AW23a] Fernando Argentieri and Hao Wu. *HS 2023 - MAT 115 Foundation of Mathematics Problem sheet 0*. 2023.

[AW23b] Fernando Argentieri and Hao Wu. *HS 2023 - MAT 115 Foundation of Mathematics Problem sheet 2*. 2023.

[AW23c] Fernando Argentieri and Hao Wu. *HS 2023 - MAT 115 Foundation of Mathematics Problem sheet 4*. 2023.

[AW23d] Fernando Argentieri and Hao Wu. *HS 2023 - MAT 115 Foundation of Mathematics Problem sheet 6*. 2023.

[BE23] Kevin Buzzard and Jon Eugster. *Natural Number Game*. 2023. URL: `https://adam.math.hhu.de/#/g/leanprover-community/nng4` (visited on 15/01/2024).

[Bot23] Mattia Luciano Bottoni. *Lean-meetings*. 2023. URL: `https://github.com/MattiaBottoni/Lean-meetings` (visited on 06/05/2024).

[BP21] Kevin Buzzard and Mohammad Pedramfar. *The Natural Number Game, version 1.3.3*. 2021. URL: `https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/index2.html` (visited on 29/01/2024).

[Buz22] Kevin Buzzard. *Formalising Mathematics*. 2022.

[CMP14] Claudio Caduff, Walter Mahler and Daniel Plüss. *Unterrichten an Berufsfachschulen [...] Berufsmaturität / Claudio Caduff ... Beitr. von Elisabeth Zillig ..* In collab. with Claudio Caduff. 2. überarb., aktualisierte und erw. Aufl. OCLC: 892464635. Bern: hep, der Bildungsverlag, 2014. ISBN: 978-3-0355-0191-9.

[Com18] Lean Community. *Zulip chat*. 2018. URL: `https://leanprover.zulipchat.com` (visited on 29/01/2024).

[Com23a] Lean Community. *Lean Manual - Setting up Lean*. 2023. URL: `https://lean-lang.org/lean4/doc/quickstart.html` (visited on 27/04/2024).

[Com23b] Lean Community. *Mathematicians' favorite proof assistant?* 2023. URL: `https://leanprover.zulipchat.com/#narrow/stream/113488-general/topic/mathematicians'.20favorite.20proof.20assistant.3F` (visited on 27/04/2024).

[Com24] Lean Community. *Advertising a course*. 2024. URL: `https://leanprover.zulipchat.com/#narrow/stream/187764-Lean-for-teaching/topic/Advertising.20a.20course/near/413129683` (visited on 27/04/2024).

[DAT24a] Team DATAtab. *DATAtab: Online Statistics Calculator*. 2024. URL: `https://datatab.de/tutorial/mann-whitney-u-test` (visited on 30/05/2024).

[DAT24b] Team DATAtab. *DATAtab: Online Statistics Calculator*. 2024. URL: `https://datatab.net/tutorial/t-distribution` (visited on 28/05/2024).

[DAT24c] Team DATAtab. *t-Test - Full Course - Everything you need to know.* 2024. URL: `https://www.youtube.com/watch?v=VekJxtk4BYM&t=6s` (visited on 28/05/2024).

[DR93] Edward L Deci and Richard M Ryan. 'Die Selbstbestimmungstheorie der Motivation und ihre Bedeutung für die Pädagogik'. In: *Zeitschrift für Pädagogik* 39.2 (1993), pp. 223–238.

[EW02] Jacquelynne S Eccles and Allan Wigfield. 'Motivational beliefs, values, and goals'. In: *Annual review of psychology* 53.1 (2002), pp. 109–132.

[FBC17] Alex Freeman, Samantha Adams Becker and Michele Cummins. *NMC/-CoSN horizon report: 2017 K.* Tech. rep. The New Media Consortium, 2017.

[FBL73] Abraham Adolf Fraenkel, Yehoshua Bar-Hillel and Azriel Levy. *Foundations of set theory.* Elsevier, 1973.

[FF10] Karl Frey and Angela Frey-Eiling. *Ausgewählte Methoden der Didaktik.* vdf Hochschulverlag AG an der ETH Zürich, 2010.

[Fur23] Marius Furter. *Logic and Foundations with Haskell.* 2023. URL: `https://www.youtube.com/watch?v=0HIm0-me_sg&list=PLd8NbPjkXPliojM8YMN3z3o9--zXwti8Z` (visited on 27/04/2024).

[Geu09] Herman Geuvers. 'Proof assistants: History, ideas and future'. In: *Sadhana* 34 (2009), pp. 3–25.

[HA+16] Conrad Hughes, Clementina Acedo et al. *Guiding Principles for Learning in the Twentyfirst Century.* 2016.

[Ham18] Richard H. Hammack. *Book of proof.* Third edition, edition 3.1. OCLC: 1112171255. Richmond, Virginia: Richard Hammack, 2018. ISBN: 978-0-9894721-2-8.

[HS21] Sarah Hofer and Lennart Schalk. 'Das individuelle Lernen unterstützen–Formative Assessments'. In: *Professionelles Handlungswissen für Lehrerinnen und Lehrer: lernen–lehren–können. Kohlhammer-Verlag: Stuttgart. S* (2021), pp. 117–133.

[Mag78] Robert Frank Mager. *Lernziele und Unterricht.* Beltz, 1978.

[Mey20] Hilbert Meyer. *Leitfaden Unterrichtsvorbereitung.* 10. Auflage. Berlin: Cornelsen, 2020. 256 pp. ISBN: 978-3-589-22458-6.

[Mey21] Hilbert Meyer. *Was ist guter Unterricht?* 15. Auflage. Berlin: Cornelsen, 2021. 192 pp. ISBN: 978-3-589-22047-2.

[MW23] Scott Morrison and Eric Wieser. *Lean 3 create new project does not work.* 2023. URL: `https://leanprover.zulipchat.com/#narrow/stream/113488-general/topic/Lean.203.20create.20new.20project.20does.20not.20work` (visited on 15/01/2024).

[Pau15] Christine Paulin-Mohring. *Introduction to the calculus of inductive constructions.* 2015.

[Pau86] Lawrence C Paulson. 'Natural deduction as higher-order resolution'. In: *The Journal of Logic Programming* 3.3 (1986), pp. 237–258.

[Pet14] Dominik Petko. *Einführung in die Mediendidaktik. Lehren und Lernen mit digitalen Medien. Beltz.* 2014.

[Pet22] Dominik Petko. *Lecture Notes in Allgemeiner Didaktik - Intelligentes Üben: Aufgabenstellungen und Lernbegleitung.* 2022.

[Pro13] The Univalent Foundations Program. 'Homotopy type theory: Univalent foundations of mathematics'. In: *arXiv preprint arXiv:1308.0729* (2013).

[Ret22] Theo Reto. *Lambda Cube Unboxed.* 2022. URL: `https://www.youtube.com/watch?v=UCzE15Hvs1E&list=PLNwzBl6BGLwOKBFVbvp-GFjAA_ESZ--q4` (visited on 27/04/2024).

[Rod22] Raúl Rodríguez Momblona. *Type theory and theorem proving in Lean.* 2022.

[Rus08] Bertrand Russell. 'Mathematical logic as based on the theory of types'. In: *American journal of mathematics* 30.3 (1908), pp. 222–262.

[Sch23] Southampton Education School. *Learning Mathematics with Lean 2nd event.* 2023. URL: `https://www.youtube.com/watch?v=q8n1vqXJe1k&list=PLsTeSHxNNZbw5DHMFbDYnj_Mf74c_-pFW` (visited on 29/04/2024).

[ST15] Ian Stewart and David Orme Tall. *The foundations of mathematics.* Second Edition. Oxford: Oxford University Press, 2015. 391 pp. ISBN: 978-0-19-870643-4.

[ST22] Elif Saçikara and Ödül Tetik. *Seminar on Lean.* 2022. URL: `https://sites.google.com/view/uzh-lean-seminar/home?pli=1` (visited on 27/04/2024).

[Sta24] Jeremy Stangroom. *P Value from Z Score Calculator. Social Science Statistics.* 2024. URL: `https://www.socscistatistics.com/pvalues/normaldistribution.aspx` (visited on 30/05/2024).

[Tao23] Terence Tao. *As a consequence of my Lean4 formalization project I have found a small (but non-trivial) bug in my paper.* 2023. URL: `https://mathstodon.xyz/@tao/111287749336059662` (visited on 28/01/2024).

[TI21] Athina Thoma and Paola Iannone. 'Learning about proof with the theorem prover lean: the abundant numbers task'. In: *International Journal of Research in Undergraduate Mathematics Education* (2021), pp. 1–30.

[VC78] Lev Semenovich Vygotsky and Michael Cole. *Mind in society: Development of higher psychological processes.* Harvard university press, 1978.

[Wah13] Diethelm Wahl. *Lernumgebungen erfolgreich gestalten: vom trägen Wissen zum kompetenten Handeln.* 3. Auflage mit Methodensammlung. Bad Heilbrunn: Verlag Julius Klinkhardt, 2013. 311 pp. ISBN: 978-3-7815-1907-7.

[Wam20] Philippe Wampfler. *Schweizer Gymnasien brauchen dringend eine Total-Revision.* 2020. URL: `https://www.nzz.ch/schweiz/schweizer-gymnasien-brauchen-dringend-eine-total-revision-ld.1790806` (visited on 27/04/2024).

[Wet19] Adrian Wetzel. *Formelsammlung Mathematik.* In collab. with Adrian Wetzel. 8. Auflage 2019. OCLC: 1303912572. Basel: Adrian Wetzel, 2019. ISBN: 978-3-9523907-1-9.

[Wod14] Rita Wodzinski. 'Leistungsheterogenität im naturwissenschaftlichen Unterricht–methodische Ansätze und empirische Befunde'. In: *Heterogenität und Diversität–Vielfalt der Voraussetzungen im naturwissenschaftlichen Unterricht. Gesellschaft für Didaktik der Chemie und Physik Jahrestagung in Bremen* (2014).