Logic and Proof

Jeremy Avigad
Robert Y. Lewis
Floris van Doorn

Version 4bf1dbe, updated at 2015-08-19 23:28:05 -0400

Contents

1

Introduction

1.1 Mathematical Proof

Although there is written evidence of mathematical activity in Egypt as early as 3000 BC, many scholars locate the birth of mathematics proper in ancient Greece around the sixth century BC, when deductive proof was first introduced. Aristotle credited Thales of Miletus with recognizing the importance of not just what we know but how we know it, and finding the appropriate grounds for knowledge via the deductive method. Around 300 BC, Euclid codified a deductive approach to geometry in his treatise, the *Elements*. Through the centuries, Euclid's axiomatic style was held as a paradigm of rigorous argumentation, not just in mathematics, but in philosophy and the sciences as well.

Here is an example of an ordinary proof, in contemporary mathematical language. It establishes a fact that was known to the Pythagoreans.

Theorem. $\sqrt{2}$ is irrational, which is to say, it cannot be expressed as a fraction a/b, where a and b are integers.

Proof. Suppose $\sqrt{2} = a/b$ for some pair of integers a and b. By removing any common factors, we can assume a/b is in lowest terms, so that a and b have no factor in common. Then $a = \sqrt{2}b$, and squaring both sides, we have $a^2 = 2b^2$.

The last equation implies that a^2 is even, and since the square of an odd number is odd, a itself must be even as well. We therefore have $a = 2c$ for some integer c. Substituting this into the equation $a^2 = 2b^2$, we have $4c^2 = 2b^2$, and hence $2c^2 = b^2$. This means that b^2 is even, and so b is even as well.

The fact that a and b are both even contradicts the fact that a and b have no common factor. So the original assumption that $\sqrt{2} = a/b$ is false.

In the next example, we focus on the natural numbers,

$$\mathbb{N} = \{0, 1, 2, \ldots\}$$

A natural number n greater than or equal to 2 is said to be *composite* if it can be written as a product $n = m \cdot k$ where neither m nor k is equal to 1, and *prime* otherwise. Notice that if $n = m \cdot k$ witnesses the fact that n is composite, then m and k are both smaller than n. Notice also that, by convention, 0 and 1 are considered neither prime nor composite.

Theorem. Every natural number greater than equal to 2 can be written as a product of primes.

Proof. We proceed by induction on n. Let n be any natural number greater than 2. If n is prime, we are done; we can consider n itself as a product with one term. Otherwise, n is composite, and we can write $n = m \cdot k$ where m and k are smaller than n. By the inductive hypothesis, each of m can be written as a product of primes, say

$$m = p_1 \cdot p_2 \cdot \ldots \cdot p_u$$

and

$$k = q_1 \cdot q_2 \cdot \ldots \cdot q_v.$$

But then we have

$$n = m \cdot k = p_1 \cdot p_2 \cdot \ldots \cdot p_u \cdot q_1 \cdot q_2 \cdot \ldots \cdot q_v,$$

a product of primes, as required.

The first goal of this course is to teach you to write clear, readable mathematical proofs. We will do this by considering a number of examples, but also by taking a reflective point of view: we will carefully study the components of mathematical language and the structure of mathematical proofs, in order to gain a better understanding of how they work.

1.2 Symbolic Logic

Towards gaining a better understanding of how proofs work, it will be helpful to study a subject known as "symbolic logic," which provides an idealized model of mathematical language and proof. In the *Prior Analytics*, the ancient Greek philosopher set out to analyze patterns of reasoning, and developed the theory of the *syllogism*. Here is one instance of a syllogism:

Every man is an animal.

Every animal is mortal.

Therefore every man is mortal.

Aristotle observed that the correctness of this inference has nothing to do with the truth or falsity of the individual statements, but, rather, then general pattern:

Every A is B.
Every B is C.
Therefore every A is C.

We can substitute various properties for A, B, and C: try substituting the properties of being a fish, being a unicorn, being a swimming creature, being a mythical creature, etc. The various statements may come out true or false, but all the instantiations will have the following crucial feature: if the two hypotheses come out true, then the conclusion comes out true as well. We express this by saying that the inference is *valid*.

Although the patterns of language addressed by Aristotle's theory of reasoning are limited, we have him to thank for a crucial insight: we can classify valid patterns of inference by their logical form, while abstracting away specific content. It is this fundamental observation that underlies the entire field of symbolic logic.

In the seventeenth century, Leibniz proposed the design of a *characteristica universalis*, a universal symbolic language in which one would express any assertion in a precise way, and a *calculus ratiocinatur*, a "calculus of thought" which would express the precise rules of reasoning. Leibniz himself took some steps to develop such a language and calculus, but much greater strides were made in the nineteenth century, through the work of Boole, Frege, Peirce, Schroeder, and others. Early in the twentieth century, these efforts blossomed into the field of mathematical logic.

If you consider the examples of proofs in the last section, you will notice that some terms and rules of inference are specific to the subject matter at hand, having to do with numbers, the properties of being prime, composite, even, odd, and so on. But there are other terms and rules of inference that are not domain specific, such as those related to the words "every," "some," "and," and "if ... then." The goal of symbolic logic is to identify these core elements of reasoning and argumentation and explain how they work, as well as to explain how more domain-specific notions are introduced and used.

To that end, we will introduce symbols for key logical notions, including the following:

- $A \to B$, "if A then B"
- $A \wedge B$, "A and B"
- $A \vee B$, "A or B"
- $\neg A$, "not A"
- $\forall x\ A$, "for every x, A"

- $\exists x\ A$, “for some x, A”

We will then provide a formal proof system that will let us establish, deductively, that certain entailments between such statements are valid.

The proof system we will use is a version of *natural deduction*, a type of proof system introduced by Gerhard Gentzen in the 1930’s to model informal styles of argument. In this system, the fundamental unit of judgement is the assertion that an assertion, A, follows from a finite set of assertions, Γ. This is written as $\Gamma \vdash A$. If Γ and Δ are two finite sets of hypotheses, we will write Γ, Δ for the *union* of these two sets, that is, the set consisting of all the hypotheses in each. With these conventions, the rule for the conjunction symbol can be expressed as follows:

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B}$$

This should be interpreted as follows: assuming A follows from the hypotheses Γ, and B follows from the hypotheses Δ, $A \wedge B$ follows from the hypotheses in both Γ and Δ.

We will see that one can write such proofs more compactly leaving the hypotheses implicit, so that the rule above is expressed as follows:

$$\frac{A \qquad B}{A \wedge B}$$

In this format, a snippet of the first proof in the previous section might be rendered as follows:

$$\frac{\dfrac{\dfrac{\dfrac{}{\neg\mathsf{E}ven(b)} \qquad \dfrac{\forall x\ (\neg\mathsf{E}ven(x) \to \neg\mathsf{E}ven(x^2))}{\neg\mathsf{E}ven(b) \to \neg\mathsf{E}ven(b^2))}}{\neg\mathsf{E}ven(b^2)} \qquad \mathsf{E}ven(b^2)}{\bot}}{\mathsf{E}ven(b)}$$

The complexity of such proofs can quickly grown out of hand, and complete proofs of even elementary mathematical facts can become quite long. Such systems are not designed for writing serious mathematics. Rather, they provide idealized models of mathematical reasoning, and insofar as they capture something of the structure of an informal proof, they enable us to study the properties of mathematical reasoning.

The second goal of this course is to help you understand natural deduction, as an example of a formal deductive system.

1.3 Interactive Theorem Proving

Early work in mathematical logic aimed to show that ordinary mathematical arguments could be modeled in symbolic calculi, at least in principle. As noted above, complexity issues limit the range of what can be accomplished in practice; even elementary mathematical arguments require long derivations that are hard to write and hard to read, and do little to promote understanding of the underlying mathematics.

Since the end of the twentieth century, however, the advent of computational proof assistants has begun to make complete formalization feasible. Working interactively with theorem proving software, users can construct formal derivations of complex theorems that can be stored and checked by computer. Automated methods can be used to fill in small gaps by hand, verify long calculations axiomatically, or fill in long chains of inferences deterministically. The reach of automation is currently fairly limited, however, and the general goal is for users to present just enough information to the system to enable it to construct and check a formal derivation. This typically involves writing proofs in a sort of "programming language" that is designed with that purpose in mind. For example, here is a short proof in the *Lean* theorem prover:

```
section
variables (p q : Prop)

theorem my_theorem : p ∧ q → q ∧ p :=
assume H : p ∧ q,
have p, from and.left H,
have q, from and.right H,
show q ∧ p, from and.intro `q` `p`

end
```

If you are reading the present text in online form, you will find a button underneath the formal "proof script" that says "Try it yourself." Pressing the button copies the proof to an editor window at right, and runs a version of *Lean* inside your browser to process the proof, turn it into an axiomatic derivation, and verify its correctness. You can experiment by varying the text in the editor and pressing the "play" button to see the result.

Proofs in Lean can access a library of prior mathematical results, all verified down to axiomatic foundations. A goal of the field of interactive theorem proving is to reach the point where any contemporary theorem can be verified in this way. For example, here is a formal proof that the square root of two is irrational, following the model of the informal proof presented above:

```
import data.rat
open eq.ops nat

theorem sqrt_two_irrational {a b : ℕ} (co : coprime a b) : a^2 ≠ 2 * b^2 :=
assume H : a^2 = 2 * b^2,
have even (a^2), from even_of_exists (exists.intro _ H),
have even a, from even_of_even_pow this,
obtain c (aeq : a = 2 * c), from exists_of_even this,
have 2 * (2 * c^2) = 2 * b^2, by rewrite [-H, aeq, *pow_two, mul.assoc, mul.left_comm c],
have 2 * c^2 = b^2, from eq_of_mul_eq_mul_left dec_trivial this,
have even (b^2), from even_of_exists (exists.intro _ (eq.symm this)),
have even b, from even_of_even_pow this,
have 2 | gcd a b, from dvd_gcd (dvd_of_even `even a`) (dvd_of_even `even b`),
have 2 | 1, from co ▶ this,
absurd `2 | 1` dec_trivial
```

The third goal of this course is to teach you to write elementary proofs in *Lean*. The facts that we will ask you to prove in Lean will be more elementary than the informal proofs we will ask you to write, but our intent is that formal proofs will model and clarify the informal proof strategies we will teach.

1.4 The Semantic Point of View

As we have presented the subject here, the goal of symbolic logic is to specify a language and rules of inference that enable us to get at the truth in a reliable way. The idea is that the symbols we choose denote objects and concepts that have a fixed meaning, and the rules of inference we adopt enable us to draw true conclusions from true hypotheses.

One can adopt another view of logic, however, as a system where some symbols have a fixed meaning, such as the symbols for "and," "or," and "not," and others have a meaning that is taken to vary. For example, the expression $P \wedge (Q \vee R)$, read "P and either Q or R," may be true or false *depending on the basic assertions that P, Q, and R*; more specifically, on their truth values. For example, if P, Q, and R stand for "seven is prime," "seven is even," and "seven is odd," respectively, then the expression is true. If we replace "seven" by "six," the statement is false. More generally, the expression comes out true whenever P is true and at least one of Q and R is true, and false otherwise.

From this perspective, logic is not so much a language for asserting truth, but a language for describing possible states of affairs. In other words, logic provides a specification language, with expressions that can be true or false depending on how we interpret the symbols that are allowed to vary. For example, if we fix the meaning of the basic predicates, the statement "there is a red block between two blue blocks" may be true or false of a given "world" of blocks, and we can take the expression to pick out the set of worlds in which it is true.

Such a view of logic is important in computer science, where we use logical expressions to select entries from a database matching certain criteria, to specify properties of hardware and software systems, or to specify constraints that we would like a constraint solver to satisfy.

There are important connections between the syntactic / deductive point of view, on the one hand, and the semantic / model-theoretic point of view, on the other. We will explore some of these along the way. For example, we can view the "valid" assertions as those that are true under all possible interpretations of the non-fixed symbols, and the "valid" inferences as those that maintain truth in all possible states and affairs. From this point of view, a deductive system should only allow us to derive valid assertions and entailments, a property known as *soundness*. If a deductive system is strong enough to allow us to verify *all* valid assertions and entailments, it is said to be *complete*.

The fourth goal of course is to convey the semantic view of logic, and understand how logical expressions can be used to characterize states of affairs.

1.5 Goals Summarized

To summarize, these are the goals of this course:

- to teach you to write clear, "literate," mathematical proofs
- to introduce you to symbolic logic and the formal modeling of deductive proof
- to introduce you to interactive theorem proving
- to teach you to understand how to use logic as a precise specification language.

Let us take a moment to comment on the relationship between some of these goals. It is important not to confuse the first three. We are dealing with three different kinds of mathematical language: ordinary mathematical language, the symbolic representations of mathematical logic, and computational implementations in interactive proof assistants. These are very different things!

Symbolic logic is not meant to replace ordinary mathematical language, and you should not use symbols like $\wedge$ and $\vee$ in ordinary mathematical proofs, any more than you would use them in place of the words "and" and "or" in letters home to your parents. Natural languages provide nuances of expression that can convey levels of meaning and understanding that go beyond pattern matching to verify correctness. At the same time, modeling mathematical language with symbolic expressions provides a level of precision that makes it possible to turn mathematical language itself into an object of study. Each has its place, and we hope to get you to appreciate the value of each without confusing the two.

The proof languages used by ineractive theorem provers lie somewhere between the two extremes. On the one hand, they have to be specified with enough precision for a computer to process them and act appropriately; on the other hand, they aim to capture some of the higher-level nuances and features of informal language in a way that enables us to write more complex arguments and proofs. Rooted in symbolic logic and designed with ordinary mathematical language in mind, they aim to bridge the gap between the two.

1.6 About These Notes

Lean is a new theorem prover, and is still under development. Similarly, these notes are being written on the fly as the class proceeds. The first time through, parts will necessarily be sketchy, buggy, and incomplete. They will therefore at best serve as a supplement to class notes and the textbook, Daniel Velleman's *How to Prove it: A Structured Approach*. Please bear with us! Your feedback will be quite helpful to improving the notes.

2 Propositional Logic

2.1 A Puzzle

The following puzzle, titled “Malice and Alice” is from George J. Summers, *Logical Deduction Puzzles.*

Alice, Alice’s husband, their son, their daughter, and Alice’s brother were involved in a murder. One of the five killed one of the other four. The following facts refer to the five people mentioned:

1. A man and a woman were together in a bar at the time of the murder.
2. The victim and the killer were together on a beach at the time of the murder.
3. One of Alice’s children was alone at the time of the murder.
4. Alice and her husband were not together at the time of the murder.
5. The victim’s twin was not the killer.
6. The killer was younger than the victim.

Which one of the five was the victim?

Take the time to try to work out a solution. Summers’ book offers the following hint: “First find the locations of two pairs of people at the time of the murder, and then determine who the killer and the victim were so that no condition is contradicted.”

2.2 A Solution

If you have worked on the puzzle, you may have noticed a few things. First, it is helpful to draw a diagram, and be systematic about searching for an answer. The number of characters, locations, and attributes (murderer, victim) is finite, so that there are only finitely many possible “states of affairs” that need to be considered. The numbers are also small enough so that systematic search through all the possibilities, though tedious, will eventually get you to the right answer.

Another thing that you may have noticed is that the question seems to presuppose that there is a unique answer to the question, which is to say, of all the states of affairs that meet the list of conditions, there is only one person who can possibly be the killer. *A priori*, without that assumption, there is a difference between finding *some* person who could have been the victim, and show that that person *had* to be the victim. Thus, there is a difference between exhibiting some state of affairs that meets the criteria, and proving that one of the characters *had* to be the victim, which is to say, demonstrating conclusively that no other solution is possible.

The published solution in the book not only produces a state of affairs that meets the criterion, but at the same time proves that this is the only one that does so. It is quoted below, in full.

From `[1]`, `[2]`, and `[3]`, the roles of the five people were as follows: Man and Woman in the bar, Killer and Victim on the beach, and Child alone.

Then, from `[4]`, either Alice’s husband was in the bar and Alice was on the beach, or Alice was in the bar and Alice’s husband was on the beach.

If Alice’s husband was in the bar, the woman he was with was his daughter, the child who was alone was his son, and Alice and her brother were on the beach. Then either Alice or her brother was the victim; so the other was the killer. But, from `[5]`, the victim had a twin, and this twin was innocent. Since Alice and her brother could only be twins to each other, this situation is impossible. Therefore Alice’s husband was not in the bar.

So Alice was in the bar. If Alice was in the bar, she was with her brother or her son.

If Alice was with her brother, her husband was on the beach with one of the two children. From `[5]`, the victim could not be her husband, because none of the others could be his twin; so the killer was her husband and the victim was the child he was with. But this situation is impossible, because it contradicts `[6]`. Therefore, Alice was not with her brother in the bar.

So Alice was with her son in the bar. Then the child who was alone was her daughter. Therefore, Alice’s husband was with Alice’s brother on the beach. From previous reasoning, the victim could not be Alice’s husband. But the victim could be Alice’s brother because Alice could be his twin.

So *Alice’s brother was the victim* and Alice’s husband was the killer.

This argument relies on some "extra logical" elements, for example, that a father cannot be younger than his child, and that a parent and his or her child cannot be twins. But there are also a number of common logical terms, and associated patterns of inference. In the next section, we will focus on some of the rules governing the terms "and," "or," "not," and "if ... then". Following the model described in the introduction, each such construction will be analyzed on three levels:

- the way it is used and employed in informal (mathematical) arguments
- a formal, symbolic representation
- the implementation in *Lean*

2.3 Rules of Inference

Implication

The first pattern of reasoning we will discuss, involving "if ... then," may be the most confusing, because its use is largely implicit in the solution above. Consider the fourth paragraph. Spelled out in greater detail, the inference runs as follows:

If Alice was in the bar, Alice was with her brother or son.
Alice was in the bar.
Alice was with her brother or son.

This rule is sometimes known as *modus ponens*, or "implication elimination," since it tells us how to use an implication in an argument. In a system of natural deduction, it is expressed as follows:

$$\frac{A \to B \qquad A}{B} \to\mathrm{E}$$

The way to read this is as follows: if you have a proof of $A \to B$, possibly from some hypotheses, and a proof of A, possibly from hypotheses, then combining these yields a proof of B, from the hypotheses in both subproofs.

In Lean, the inference is expressed as follow:

```
variables (A B : Prop)
premises (H₁ : A → B) (H₂ : A)

example : B :=
show B, from H₁ H₂
```

The first command declares two variables, `A` and `B`, ranging over propositions. The second line introduces two premises, namely, `A → B` and `A`. The next line asserts, as an example, that `B` follows from the premises. The proof is written simply $\mathtt{H}_1$ $\mathtt{H}_2$: think of this as the premise $\mathtt{H}_1$ "applied to" the premise $\mathtt{H}_2$.

You can enter the arrow by writing `\to` or `\imp` or `\r`. You can enter $\mathtt{H}_1$ by typing `H\_1`. It is conventional to use the letter `H` for a hypothesis, but you can use any reasonable alphanumeric identifier. The name `H1` is a different from $\mathtt{H}_1$, but also a reasonable choice.

The rule for proving an "if . . . then" statement is more subtle. Consider the beginning of the third paragraph, which argues that if Alice's husband was in the bar, then Alice or her brother was the victim. Abstracting away some of the details, the argument has the following form:

Suppose Alice's husband was in the bar.
  Then . . .
  Then . . .
  Then Alice or her brother was the victim.
  Thus, if Alice's husband was in the bar, then Alice or her brother was the victim.

This is a form of *hypothetical reasoning*. On the supposition that A holds, we argue that B holds as well. If we have successful, we have shown that A implies B, without supposing A. In other words, the temporary assumption that A holds is "canceled" by making it explicit in the conclusion.

$$\dfrac{\begin{array}{c}\overline{H : A}\\ \vdots \\ \psi\end{array}}{A \to B}\ \to\!\mathrm{I}, H$$

The hypothesis is given the label H; when the introduction rule is applied, the label H indicates the relevant hypothesis. The line over the hypothesis indicates that the assumption has been "canceled" by the introduction rule.

In Lean, this inference takes the following form:

```
variables (A B : Prop)

example : A → B :=
assume H : A,
show B, from sorry
```

To prove `A → B`, we assume `A`, with label `H`, and show `B`. Here, the word `sorry` indicates that the proof is omitted. In this case, this is necessary; since `A` and `B` are arbitrary propositions, there is no way to prove `B` from `A`. In general, though, `A` and `B` will be compound expressions, and you are free to use the hypothesis `H : A` to prove `B`.

Using `sorry`, we can illustrate the implication elimination rule alternatively as follows:

```
variables (A B : Prop)

example : B :=
have H₁ : A → B, from sorry,
have H₂ : A, from sorry,
show B, from H₁ H₂
```

Conjunction

As was the case for implication, other logical connectives are generally characterized by their *introduction* and *elimination* rules. The former show how to establish a claim involving the connective, while the latter show how to use such a claim to derive others.

Let us consider, for example, the case of conjunction, that is, the word "and." Informally, we establish a conjunction by establishing each conjunct. For example, informally we might argue:

Alice's brother was the victim.

Alice's husband was the killer.

Therefore Alice's brother was the victim and Alice's husband was the killer.

The inference seems superfluous, since the word "and" simply combines the two assertions into one, and informal proofs often downplay the distinction. In natural deduction, the rule reads as follows:

$$\frac{A \qquad B}{A \wedge B}\ \wedge\mathrm{I}$$

In Lean, the rule is denoted `and.intro`:

```
variables (A B : Prop)

example : A ∧ B :=
have H₁ : A, from sorry,
have H₂ : B, from sorry,
show A ∧ B, from and.intro H₁ H₂
```

You can enter the wedge symbol by typing `\and`.

The two elimination rules allow us to extract the two components:

Alice's husband was in the bar and Alice was on the beach.

So Alice's husband was in the bar.

Or:

Alice's husband was in the bar and Alice was on the beach.

So Alice's was on the beach.

In natural deduction, these patterns are rendered as follows:

$$\frac{A \wedge B}{A} \wedge\mathrm{E}_1 \qquad \frac{A \wedge B}{B} \wedge\mathrm{E}_2$$

In Lean, the inferences are known as `and.left` and `and.right`:

```
variables (A B : Prop)

example : A :=
have H : A ∧ B, from sorry,
show A, from and.left H

example : B :=
have H : A ∧ B, from sorry,
show B, from and.right H
```

Negation and Falsity

In logical terms, showing "not A" amounts to showing that A leads to a contradiction. For example:

Suppose Alice's husband was in the bar.

. . .

This situation is impossible.

Therefore Alice's husband was not in the bar.

This is another form of hypothetical reasoning, similar to that used in establishing an "if . . . then" statement: we temporarily assume A, show that leads to a contradiction, and conclude that "not A" holds.

In natural deduction, the rule reads as follows:

$$\dfrac{\dfrac{\overline{A}}{\vdots}\\ \bot}{\neg A}\neg\mathrm{I}$$

In Lean, it is illustrated by the following:

```
variable A : Prop
```

```
example : ¬ A :=
assume H : A,
show false, from sorry
```

You can enter the negation symbol by typing `\not`.

The elimination rule is dual to these. It expresses that if we have both “A” and “not A,” then we have a contradiction. This pattern is illustrated in the informal argument below, which is implicit in the fourth paragraph of the solution to “Malice and Alice.”

So the killer was her husband and the victim was the child he was with.

So the killer was not younger than his victim.

But according to `[6]`, the killer was younger than his victim.

This situation is impossible.

In symbolic logic, the rule of inference is expressed as follows:

$$\frac{\neg A \qquad A}{\bot}\ \neg\mathrm{E}$$

And in Lean, it is implemented in the following way:

```
variable A : Prop

example : false :=
have H₁ : ¬ A, from sorry,
have H₂ : A, from sorry,
show false, from H₁ H₂
```

Notice that the negation elimination rule is expressed in a manner similar to implication elimination: the label asserting the negation comes first, and by “applying” the proof of the negation to the proof of the positive fact, we obtain a proof of falsity.

Notice that in the symbolic framework, we have introduced a new symbol, $\bot$. It corresponds to the identifier `false` in Lean, and natural language phrases like “this is a contradiction” or “this is impossible”.

What are the rules governing $\bot$? In natural deduction, there is no introduction rule; “false” is false, and there should be no way to prove it, other than extract it from contradictory hypotheses. On the other hand, natural deduction provides a rule that allows us to conclude anything from a contradiction:

$$\frac{\bot}{A}\ \bot\mathrm{E}$$

The elimination rule also has the fancy Latin name, *ex falso sequitur quodlibet*, which means “anything you want follows from falsity.” In Lean it is implemented as follows:

```
variable A : Prop

example : A :=
have H : false, from sorry,
show A, from false.elim H
```

The false elimination rule is harder to motivate from a natural language perspective, but, nonetheless, it is really needed to capture common patterns of inference. One way to understand it is this. Consider the following statement:

For every natural number n, if n is prime and greater than 2, then n is odd.

We would like to say that this is a true statement. But if it is true, then it is true of any number n. Taking $n = 2$, we have the statement:

If 2 is prime and greater than 2, then 2 is odd.

In this conditional statement, both the antecedent and succedent are false. The fact that we are committed to saying that this statement is true shows that we should be able to prove, one way or another, that the statement 2 is odd follows from the false statement that 2 is prime and greater than 2. The *ex falso* neatly encapsulates this sort of inference.

Notice that if we define $\neg A$ to be $A \to \bot$, then the rules for negation introduction and elimination are nothing more than implication introduction and elimination, respectively. We have think of $\neg A$ expressed colorfully by saying "if A is true, then pigs have wings," where "pigs have wings" is stands for $\bot$.

Disjunction

The introduction rules for disjunction, otherwise known as "or," are straightforward. For example, the claim that condition `[3]` is met in the proposed solution can be justified as follows:

Alice's daughter was alone at the time of the murder.

Therefore, either Alice's daughter was alone at the time of the murder, or Alice's son was alone at the time of the murder.

In terms of natural deduction, the two introduction rules are as follows:

$$\frac{A}{A \vee B}\ \vee\mathrm{I_l} \qquad \frac{B}{A \vee B}\ \vee\mathrm{I_r}$$

Here, the l and r stand for "left" and "right". In Lean, they are implemented as follows:

```
variables (A B : Prop)

example : A ∨ B :=
have H : A, from sorry,
show A ∨ B, from or.inl H
```

You can enter the vee symbol by typing `\or`. The identifiers `inl` and `inr` stand for "insert left" and "insert right," respectively.

The disjunction elimination rule is trickier, but it represents a natural form of case-based hypothetical reasoning. The instances that occur in the solution to "Malice and Alice" are all special cases of this rule, so it will be helpful to make up a new example. Suppose, in the argument above, we had established that either Alice's brother or her son was in the bar, and we wanted to argue for the conclusion that her husband was on the beach. One option is to argue by cases: first, consider the case that her brother was in the bar, and argue for the conclusion on the basis of that assumption; then consider the case that her son was in the bar, and argue for the same conclusion, this time on the basis of the second assumption. Since the two cases are exhaustive, if we know that the conclusion holds in each case, we know that it holds outright. The pattern looks something like this:

Either Alice's brother was in the bar, or Alice's son was in the bar.

Suppose, in the first case, that her brother was in the bar. Then … Therefore, her husband was on the beach.

On the other hand, suppose her son was in the bar. In that case, … Therefore, in this case also, her husband was on the beach.

Either way, we have established that her husband was on the beach.

In natural deduction, this pattern is expressed as follows:

$$\dfrac{A \vee B \qquad \begin{array}{c}\overline{A}\\ \vdots\\ C\end{array} \qquad \begin{array}{c}\overline{B}\\ \vdots\\ C\end{array}}{C}\;\vee\mathrm{E}$$

And here it is in Lean:

```
variables (A B C : Prop)

example : C :=
have H : A ∨ B, from sorry,
show C, from or.elim H
  (assume H₁ : A,
    show C, from sorry)
  (assume H₂ : B,
    show C, from sorry)
```

What makes this pattern confusing is that it requires to instances of nested hypothetial reasoning: in the first block of parentheses, we temporarily assume `A`, and in the second block, we temporarily assume `B`. When the dust settles, we have established `C` outright.

Bi-implication

"If and only if"

True

Proof by Contradiction

2.4 Writing Proofs in Natural Deduction

As noted in Chapter Introduction, there are two common styles for writing natural deduction derivations. (The word "derivation" is often used to connote a formal proof instead of an informal one. When talking about natural deduction, we will use the words "derivation" and "proof" interchangeably.) In both cases, proofs are presented on paper as trees, with the conclusion at the theorem at the root, and hypotheses up at the leaves. In the first style of presentation, the set of hypotheses is written explicitly at every node of the tree. This is helpful because some rules (namely, implication introduction, negation introduction, or elimination, and proof by contradiction) change the set of hypotheses, by canceling a local or temporary assumption. Nonetheless, we will use a style of presentation that leaves this information implicit, so that each node of the tree is labelled with an explicit formula. Some people like to label each inference with the rule that is used, but that is usually clear from the context, so we will omit that as well. But when a rule cancels a hypothesis, we will make that clear in the following way: we will label all instances of the hypothesis at the leaves with a letter, like "x," and then we will use that letter to annotate the place where the rule is canceled.

In addition to all the rules listed in the last section, there is one additional rule that is central to the system, namely the assumption rule. It works like this: at any point, you can assume a hypothesis, A. The way to "read" such a one-line proof is this: assuming A, you have proved A. Without this rule, there would be no way of getting a proof of the ground! After all, every rule listed in the last section has premises, which is to say, it can only be applied to derivations that have been constructed previously.

Let us consider a few examples. In each case, you should think about what the formulas say and which rule of inference is invoked at each step. Also pay close attention to which hypotheses are canceled at each stage. If you look at any node of the tree, what has been established at that point is that the claim follows from the uncanceled hypotheses. Here is a proof of $(A \wedge (B \vee C)) \to ((A \wedge B) \vee (A \wedge C))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : A \wedge (B \vee C)}}{B \vee C} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y : A \wedge (B \vee C)}}{A} \quad \dfrac{}{x : B}}{A \wedge B}}{(A \wedge B) \vee (A \wedge C)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y : A \wedge (B \vee C)}}{A} \quad \dfrac{}{x : C}}{A \wedge C}}{(A \wedge B) \vee (A \wedge C)}}{(A \wedge B) \vee (A \wedge C)}\,{\scriptstyle x}}{(A \wedge (B \vee C)) \to ((A \wedge B) \vee (A \wedge C))}\,{\scriptstyle y}$$

There is a general heuristic that is useful for deriving theorems like these, namely:

1. First, work backwards from the conclusion, using the introduction rules. For example, if you are trying to prove a statement of the form $A \to B$, add A to your list of hypotheses and try to derive B. If you are trying to prove a statement of the form $A \wedge B$, use the and-introduction rule to reduce your task to proving A, and then proving B.

2. When you have run out things to do in the first step, use elimination rules to work forwards. If you have hypotheses $A \to B$ and A, apply modus ponens to derive B. If you have a hypothesis $A \vee B$, use or elimination and try to prove any open goals by splitting on cases, considering A in one case and B in the other.

3. If all else fails, use a proof by contradiction.

When writing expressions in symbolic logic, we will adopt the an order of operations, which allow us to drop superfluous parentheses. When parsing an expression: – negation binds most tightly – then conjunctions and disjunctions, from right to left – and finally implications and bi-implications. So, for example, the expression $\neg A \vee B \to C \wedge D$ is understood as $((\neg A) \vee B) \to (C \wedge D)$

The next proof shows that if a conclusion, C, follows from A and B, then it follows from their conjunction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to (B \to C)}\,{\scriptstyle y} \quad \dfrac{\dfrac{}{A \wedge B}\,{\scriptstyle x}}{A}}{B \to C} \qquad \dfrac{\dfrac{}{A \wedge B}\,{\scriptstyle x}}{B}}{C}}{A \wedge B \to C}\,{\scriptstyle x}}{(A \to (B \to C)) \to (A \wedge B \to C)}\,{\scriptstyle y}$$

The conclusion of the next proof can be interpreted as saying that if it is not the case that one of A or B is true, then they are both false.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A\vee B)}\,z \qquad \dfrac{\dfrac{}{A}\,x}{A\vee B}}{\bot}}{\neg A}\,x \qquad \dfrac{\dfrac{\dfrac{}{\neg(A\vee B)}\,z \qquad \dfrac{\dfrac{}{B}\,y}{A\vee B}}{\bot}}{\neg B}\,y}{\neg A\wedge\neg B}}{\neg(A\vee B)\to\neg A\wedge\neg B}\,z$$

2.5 Writing Proofs in Lean

We will see that Lean has mechanisms for modeling proofs at a higher level than natural deduction derivations. At the same time, you can also carry out low-level inferences, and carry out proofs that mirror natural deduction proofs quite closely. Here is a Lean representation of the first example in the previous section:

```
variables (A B C : Prop)

example : A ∧ (B ∨ C) → (A ∧ B) ∨ (A ∧ C) :=
assume H₁ : A ∧ (B ∨ C),
have H₂ : A, from and.left H₁,
have H₃ : B ∨ C, from and.right H₁,
show (A ∧ B) ∨ (A ∧ C), from
  or.elim H₃
    (assume H₄ : B,
      have H₅ : A ∧ B, from and.intro H₂ H₄,
      show (A ∧ B) ∨ (A ∧ C), from or.inl H₅)
    (assume H₄ : C,
      have H₅ : A ∧ C, from and.intro H₂ H₄,
      show (A ∧ B) ∨ (A ∧ C), from or.inr H₅)
```

The first line declares propositional variables `A`, `B`, and `C`. The line that begins with the keyword `example` declares the theorem to be proved, and the notation `:=` indicates that the proof will follow. The line breaks and indentation is only for the purposes of readability; Lean would do just was well if the entire proof were written as one run-on line.

Here are some additional notes:

– It is often important to name a theorem for future proof. Lean allows us to do that, using one of the keywords `theorem`, `lemma`, `proposition`, `corollary`, followed by the name of the proof.

– You can omit a label in a `have` statement. You can then refer to that fact using the label `this`, until the next anonymoyus `have`. Alternatively, at any point later in the proof, you can refer to the fact by putting the assertion between backticks.

– One can also omit the label in an `assumption` by using the keyword `suppose` instead.

With these features, the previous proof can be written as follows:

```
variables (A B C : Prop)
```

```
theorem my_theorem : A ∧ (B ∨ C) → (A ∧ B) ∨ (A ∧ C) :=
assume H₁ : A ∧ (B ∨ C),
have A, from and.left H₁,
have B ∨ C, from and.right H₁,
show (A ∧ B) ∨ (A ∧ C), from
  or.elim ‘B ∨ C‘
    (suppose B,
      have A ∧ B, from and.intro ‘A‘ ‘B‘,
      show (A ∧ B) ∨ (A ∧ C), from or.inl this)
    (suppose C,
      have A ∧ C, from and.intro ‘A‘ ‘C‘,
      show (A ∧ B) ∨ (A ∧ C), from or.inr this)
```

In fact, such a presentation provides Lean with more information than is really necessary to construct an axiomatic proof. The word `assume` can be replaced by the symbol λ, assertions can be omitted from an `assume` when they can be inferred from context, the justification of a have statement can be inserted in places where the label was otherwise used, and one can omit the `show` clauses, giving only the justification. As a result, the previous proof can be written in an extremely abbreviated form:

```
variables (A B C : Prop)

example : A ∧ (B ∨ C) → (A ∧ B) ∨ (A ∧ C) :=
λ H₁, or.elim (and.right H₁)
  (λ H₄, or.inl (and.intro (and.left H₁) H₄))
  (λ H₄, or.inr (and.intro (and.left H₁) H₄))
```

Such proofs tend to be harder to write, read, understand, maintain, and debug, however, and so we will tend to favor structure and readability over brevity.

The next proof in the previous section can be rendered in Lean as follows:

```
variables (A B C : Prop)

example : (A → (B → C)) → (A ∧ B → C) :=
assume H₁ : A → B → C,
assume H₂ : A ∧ B,
show C, from H₁ (and.left H₂) (and.right H₂)
```

And the last proof can be rendered as follows:

```
variables (A B C : Prop)

example : ¬ (A ∨ B) → ¬ A ∧ ¬ B :=
assume H : ¬ (A ∨ B),
have ¬ A, from
  suppose A,
  have A ∨ B, from or.inl ‘A‘,
  show false, from H this,
have ¬ B, from
  suppose B,
  have A ∨ B, from or.inr `B`,
  show false, from H this,
show ¬ A ∧ ¬ B, from and.intro `¬ A` `¬ B`
```

2.6 Writing Informal Proofs

2.7 Derived Rules and Coarser Steps

In the examples above, we showed that, given $A \vee B$ and $\neg A$, we can derive B in natural deduction. This is a common pattern of inference, and, having justified it once, one might reasonably want to use it freely as a new one-step inference. Similarly, having proved $A \to B$ equivalent to $\neg A \vee B$, or $\neg(A \vee B)$ equivalent to $\neg A \wedge \neg B$, one might feel justified in replacing one by the other in any expression.

Indeed, this is how informal mathematics works: even if we start with some basic rules of inference, we learn to recognize more complex patterns and apply them freely. A single step in the informal argument in the solution to "Malice and Alice," or any mathematical proof, usually requires many more steps in a formal calculus. Moreover, in ordinary mathematics, one we prove a proposition or theorem, we can freely invoke it in another proof later on.

In symbolic logic, "derived rules".

In Lean, naming theorems and reusing them. (Also, eventually, automation.)

To summarize:

– When we ask you to prove something in natural deduction, our goal is to make you work with the precise, formal rules of the system. So you should not appeal to external rules unless we explicitly say so.

– When writing informal proofs, it is a judgment call as to what prior patterns of reasoning and background facts you may appeal to. In a classroom setting, the goal may be to demonstrate mastery of the subject to the instructors, in which case, context should dictate what is allowable (and it is always a good idea to err on the side of caution). In real life, your goal is to convince your target audience, and you will have to rely on convention and experience to judge what patterns of inference you can put forth, and how much detail you need to use.

– In interactive theorem proving, the main goal is to have the computer certify the proof as correct, and in that respect, automation and facts from the library are fair game. In this class, we will try to be explicit about what we would like you to use in the exercises we assign.

2.8 Truth Tables and Semantics

In the previous sections, we've seen how to prove logical formulas from hypotheses. If the hypotheses are true, the derived formula must be true as well. A formula that can be derived from no hypotheses is said to be *valid*: for example, $A \to A$ is true no matter what we suppose about A.

Not every sentence is valid, of course. Earlier we saw the example $A \to B$. We cannot derive this formula without some extra assumptions about A and B. Try it:

```
variables A B : Prop

example : A → B :=
assume H : A,
show B, from sorry
```

Without some more information, there's no argument we could put in place of the "sorry" to complete this proof. After all, B could be false!

What do we mean by "false," exactly? Notice that we've been careful not to use the words "true" and "false" before now. Deductions and formal proofs are *syntactic* ideas- that is, they have to do with the symbols and symbolic structure of the formulas involved. Truth is a *semantic* notion- it ascribes some extra-logical *meaning* to they symbols involved.

Syntactically, we were able to ask and answer questions like the following:

- Can I derive a certain formula from certain hypotheses?
- How do I derive a certain formula from certain hypotheses?
- What formulas can I derive from certain hypotheses?

The questions we consider semantically are slightly different:

- Given a truth assignment for propositional variables, is a certain formula true or false?
- Under what conditions is a certain formula true or false?

Our notions of syntax and semantics have developed carefully, so that they exist in harmony. We'll expand on this in the next section. For now, we'll discuss the basic methods we use to answer semantic questions.

The first notion we'll need is that of a *truth value*. Conveniently, we know our two truth values already: they're just "true" and "false." (We'll use the symbols $\top$ and $\bot$, respectively.) (Jeremy, will we? Do you prefer other notation?)

In this class, we'll adopt a "classical" notion of truth. This notion comes with many implications. For now, though, it means only the following: any proposition is either true or false, but not both. This means a proposition cannot be neither true nor false.

This binary conception of truth corresponds to the syntactic tautology $A \vee \not A$. Semantically, we read this sentence as saying “either A is true, or $\not A$ is true.” Since $\not A$ is true exactly when A is false, it equivalently says “either A is true, or A is false.”

The next notion we’ll need is that of a *truth assignment*. A truth assignment simply tells us which atomic statements are true, and which are false. In formal logic, this amounts to a mapping from our propositional letters A, B, etc. to the set of truth values $\{\top, \bot\}$. For instance,

- $A := \top$
- $B := \top$
- $C := \bot$

$\vdots$

is a (partial) truth assignment.

2.9 A Complete Proof System

2.10 Exercises

Bibliography