1 Mathematical rigor vs. scientific rigor

Did the sun rise today? Did it rise every day of your life so far? Your answers to these two questions are probably a resounding ‘yes’, because we tend to trust what we observe. This kind of empiricism - basing our notion of what is true on what we observe in the natural world - is the basis for most of modern science.

Scientists would go one step further, and ask: Will the sun rise tomorrow? They might say yes, based on the fact that it has risen every morning so far. However, this does not provide a convincing argument that it will rise tomorrow. On the day before your 20th birthday, you will have been less than 20 years old on every day before that, and yet that will no longer be true on the next day!

In order to provide more conclusive proof of their claims, scientists are forced to bring in mathematics, which is the only setting in which we can be 100% certain of an answer or claim, given agreed-upon underlying assumptions. We can come up with simple mathematical models (such as Newton’s laws of gravitation) that simultaneously explain many of our observations, and then use these models to predict that the sun will indeed rise again tomorrow.

In this class, we will be focusing on the part that comes after making the mathematical models. That is, given a mathematical system, what methods can we use to draw conclusions with 100% certainty? This is the art of mathematical proof.

2 What is a proof?

Simply stated

A proof is an explanation of why a statement is objectively correct.

Thus we have two goals for our proofs: first we want to verify that it is objectively correct, and second we want to be able to most effectively and elegantly explain that to our audience.

The Proof Spectrum: Rigor vs. Elegance

These two goals are sometimes in conflict. In order to be absolutely certain our proof is correct, we need to be exceedingly careful and rigorous. In order to be clear in our exposition we need to be succinct and elegant. It’s difficult to do both simultaneously!

On the one hand mathematical proofs need to be rigorous - we naturally want our proofs to be correct. One way to ensure they are correct is to have them checked by a computer. (Note that checking to see if a proof is correct is much easier for a computer to do than finding a proof in the first place.) Proofs that can be checked by a computer are often called formal proofs.

On the other hand, most mathematicians are attracted to mathematics because of its intrinsic beauty. A proof that communicates the key ideas of a proof to the reader in a succinct and beautiful way is very effective for its expository properties, even if it is not as rigorous as a formal proof. Such a proof is called a traditional proof. The legendary mathematician Paul Erdős always spoke of “The Book”, an imaginary book in which God had written down the best and most elegant proofs for mathematical theorems. When he saw any particularly inspiring proof he would exclaim “That proof is from ‘The Book’!"

We will strive for both rigor and elegance in our proofs by studying both highly rigorous formal proofs and more elegant traditional proofs as we proceed through the course.

1These notes are adapted primarily from materials created for Prove it! Math Academy, an advanced summer math program for high school students focused on the transition to abstract mathematics and proof writing.
Formal Proof Systems

Definition 1. A Formal Proof System (or Formal Axiom System) consists of

1. A set of expressions $S$, called the **statements**.
2. A set of rules $R$, called the **rules of inference**.

Each rule of inference has zero or more inputs called **premises** and one or more outputs called **conclusions**. Most premises and all conclusions of a rule of inference are statements in the system. There also may be conditions on when a particular rule of inference can be used.

Definition 2. An **axiom** is a conclusion of a rule of inference that has no premises.

Definition 3. A statement $Q$ in a formal axiom system is **provable from** premises $Q_1, \ldots, Q_n$ if either:

1. $Q$ is one of the premises $Q_1, \ldots, Q_n$, or
2. $Q$ is a conclusion of a rule of inference whose premises are provable from $Q_1, \ldots, Q_n$.

In particular, if $Q$ is an axiom, then $Q$ is provable from no premises at all!

Definition 4. If $Q$ follows from no premises in a formal axiom system, we say that $Q$ is **provable** in the system. A provable statement is called a **theorem**.

Definition 5. A **proof** of a statement in a formal axiom system is a sequence of applications of the rules of inference that show that the statement is a theorem in that system.

**Notation.** If $Q$ is provable from premises $P_1, \ldots, P_n$ in a formal system we can denote this symbolically as

$$P_1, \ldots, P_n \vdash Q$$

It is also commonplace to refer to such an expression as a theorem. To prove such a theorem is to give a proof of $Q$ in the same formal system where additionally the premises are ‘Given’ as axioms.

Toy Proofs

There are several examples of simple Formal Proof Systems available online at

proveitmath.org/toyproofs

**Scrambler** is a formal proof system where the statements are finite sequences of colors. The Rules of Inference are permutations of these sequences (and so have one premise and one conclusion each). The goal is to apply the Rules to show that a given sequence of colors is provable from another given sequence of colors.

**Trix Game** is a formal proof system where the statements are positive integers. There are only two Rules of Inference, both of which take a single positive integer as a premise, and return a single positive integer as their conclusion. This system illustrates a rule that has a condition on when you can use it. The goal is to show that a given positive integer is provable from the premise 1 in the system.

**Circle-Dot** is a formal proof system where the statements are just finite sequences of one or more circles and dots. This system has many of the features of actual mathematical formal axiom systems. There are five rules of inference, two of which are axioms. The goal is to prove various circle-dot strings in the system.

3 The Language of Mathematics

The toy proof systems above were just warmups - it is now time to build up the axiom systems on which mathematics is founded! In particular, we need to understand what a valid mathematical **statement** (or **proposition**) is in the language of mathematics. The building blocks of this language are described by the following terms.

---

2Other common premises are variable declarations, constant declarations, and subproofs.
Expressions and Statements

As in any language, each topic in mathematics defines certain arrangements of symbols to be valid expressions about that topic. Each expression can optionally have an associated property called its type, which describes the kinds of mathematical objects that the expression might represent.

Example 6. For example, the expression “Prove it! Math is awesome.” is a valid expression in the English language. On the other hand “ls -a” is not a valid expression in the English language, but is a valid expression in some programming languages.

Example 7. Similarly, in a high school math class \( x^2 - 1 \) might be a valid expression whose type is ‘real number’, while the expression “cos” might be a valid expression whose type is ‘function’. On the other hand, an expression like “5 \( \nrightarrow \alpha \)” probably isn’t defined to be a valid expression in any mainstream topic in mathematics.

In general, a statement is any valid expression (in any language) whose type is ‘statement’. Thus, one of the first tasks in defining (resp. learning) a new topic in mathematics is to define (resp. learn) what expressions are designated as having type ‘statement’ in that language.

Example 8. In English, statements are expressions which can be either true or false. For example, “Math 108 is awesome.” is either true or false and is therefore a statement. Expressions like “Oh, my!” or “Do you like ice cream?” are valid English expressions but are neither true nor false and therefore are not statements in English.

Example 9. In a typical math course, expressions that are either true or false, like “1 + 1 = 2”, “91 is prime”, and “\( x^2 = \cos(x) \)”, are also considered to be statements. Note that we don’t need to know if such a statement is true or false, just that it is either true or false. Similarly, expressions like “42”, “\{1, 2, 3\}”, and “\( x^3 + 1 \)” are neither true nor false and therefore are not statements.

Constants and Variables

An identifier is a name or label for something. Several identifiers can also be combined to form larger expressions called compound expressions (but identifiers are not compound expressions themselves). An identifier that refers to a unique, specific object is called a constant. An identifier that refers to a single, but unspecified, object is called a variable.

Example 10. Proper names are usually constants in the English language. For example, “Paris”, “Professor Gillespie”, and “Groot” are all intended to refer to a unique, specific thing. On the other hand, when filling out a form with blanks containing “FIRST NAME GOES HERE” or “CELL”, those identifiers can be thought of as variables that can represent any first name or any phone number that might be entered into the form.

Example 11. In standard mathematics, identifiers like \( \pi \), “1024”, “\ln”, “\(+\)” and “\( \cap \)” usually represent constants, whereas identifiers like \( x \), \( n \), \( P \), \( \alpha \), and \( a_0 \) frequently represent variables.

Example 12. In the Circle-Dot system the only statements are circle-dot strings like “••” or “•••”. All of those expressions are constants.

Both constants and variables can have a type, and in that case represent an object or expression of that type.

Example 13. In Example 10 the variable “cell” might have type ‘phone number’ and someone filling out the form might find that it rejects any expression that cannot be interpreted as a phone number.

Example 14. In the Rules of Inference for the Circle-Dot system the expressions \( W \) and \( V \) are variables of type ‘circle dot string’ as they are placeholders in the rule for an unspecified circle-dot string.

\[^3\text{In this case it is obviously true.}\]
Substitution and Lambda Expressions

We can prefix an expression $E$ to form the expression $\lambda x, E$ (or $x \mapsto E$) to indicate that all occurrences of $x$ in $E$ are a variable that represents the same unspecified object of the same type as $x$. These prefixed expressions are called lambda expressions (or anonymous functions).

Such expressions can be applied to an expression $a$ having the same type as $x$ to form a new expression, $(\lambda x, E)(a)$ which has the same type as $E$. These can be further simplified to the expression obtained by replacing all occurrences of $x$ in $E$ with $a$. If we give a name to a lambda expression, e.g., define $f$ to be $\lambda x, E$ then the expression $(\lambda x, E)(a)$ is just the usual notation for function application $f(a)$.

Example 15. In high school algebra, if $x$ is a variable of type integer then $(\lambda x, x^2 + x + 1)(3)$ simplifies to $3^2 + 3 + 1$. Similarly, $(\lambda x, x + y)(3)$ simplifies to $3 + y$ while $(\lambda y, x + y)(3)$ simplifies to $x + 3$.

Two lambda expressions are said to be equivalent if they simplify to the same or equivalent things when applied to any argument. Renaming all occurrences of $x$ in $\lambda x, E$ with a new identifier always produces a lambda expression that is equivalent to the original.

Example 16. In the previous example the lambda expression $(\lambda x, x^2 + x + 1)$ is equivalent to $(\lambda y, y^2 + y + 1)$. They both simplify to the same expression when applied to the same argument. For example, $(\lambda x, x^2 + x + 1)(3)$ and $(\lambda y, y^2 + y + 1)(3)$ both simplify to $3^2 + 3 + 1$.

Another common situation where we can simplify a lambda expression $\lambda x, E$ is when the expression $E$ does not contain $x$. In this situation $(\lambda x, E)(a)$ simplifies to just $E$ for every $a$, and thus we can say that $\lambda x, E$ simplifies to just $E$ in that case.

Example 17. The expression $(\lambda x, \cos(x))(a)$ simplifies to $\cos(a)$ for every argument $a$. Thus the expression $\cos$ can be though of as the lambda expression $(\lambda x, \cos(x))$ since they both simplify to the same expression when applied to an argument $a$.

4 Rules of Inference in Mathematics

In the Circle-Dot system Axiom A is a rule of inference that says from no premises we can conclude $\circ \bullet$. Rule 1, however, is technically not a rule of inference, but rather an infinite family of rules of inference, one for each choice of circle-dot strings we can substitute for the variables $W$ and $V$. From this perspective we can think of Rule 1 as the lambda expression,

$$\lambda W, \lambda V, (WV, VW \vdash W)$$

So that, for example, substituting $\circ$ for $W$ and $\bullet$ for $V$ produces the rule

$$(\lambda W, \lambda V, (WV, VW \vdash W))(\circ)(\bullet) = (\lambda V, (\circ V, V \circ \vdash \circ))(\bullet)$$

which allows us to conclude $\circ$ from the premises $\bullet \circ$ and $\circ \bullet$.

Most rules of inference in mathematics are more similar to Rule 1 than to Axiom A in this sense – they are really lambda expressions which generate an entire family of specific rules of inference, one for each choice of variable in the statement of the rule. Because this is so common, we usually omit the lambda prefixes, and use the convention that any variable $W$ that appears in the premises or conclusion of a rule of inference can be replaced with an expression of the same type to form a particular instance of that rule of inference.

---

4 These refer to free occurrences - see the quantifiers handout for details.
5 See footnote 2. Also no free identifier in $a$ should become bound as a result of the substitution.
6 Indeed, in precalculus they usually write $f(x) = x^3$ instead of writing $f = (\lambda x, x^3)$, but the latter is usually what they mean.
Recipe Notation for Rules of Inference

Notation. A rule of inference having premises $P_1, \ldots, P_k$ and conclusions $Q_1, \ldots, Q_n$ can be expressed in recipe notation as

- Show: $P_1$
- : 
- Show: $P_k$
- Conclude: $Q_1$
- : 
- Conclude: $Q_n$

Some rules of inference have a premise of the form

$$(P_1, \ldots, P_k \vdash Q)$$

In that case, this premise is validated by including a subproof in that proof that $Q$ can be proved from the given premises (which do not need to be justified by a rule of inference). We denote this in recipe notation as an indented ‘assume-block’ as illustrated below.

**Exercise 18.** Suppose we have a rule of inference that justifies the following.

$$P \lor Q, (P \vdash R), (Q \vdash R) \vdash R$$

where $P$, $Q$, and $R$ are any mathematical statements. Then we would express this rule in recipe notation as

- Show: $P \lor Q$
  - Assume $P$
  - Show: $R$
  - Assume $Q$
  - Show: $R$
  - Conclude: $R$

In this, everything between an Assume and the following $\leftarrow$ (the ‘end assumption’ symbol) is a subproof that demonstrates the corresponding premise in the rule of inference. We indent such assumption blocks in our proofs. Subproofs can be nested, and the level of indentation corresponds to the level of nesting. Assumptions (lines that start with Assume) do not need to be justified by a rule of inference. We sometimes say that they are 'Given'.

Note that we do include the word "Assume" in the proof itself, but not the words "Show" or "Conclude" which are just instructions to the proof author (as opposed to the reader) for how to follow the recipe for this rule of inference.

5 Natural Deduction

We now turn our attention to a formal axiom system that is based on one first formulated by Gerhard Gentzen in 1934 as a formal system that closely imitates the way mathematicians actually reason when writing traditional expository proofs.
Propositional Logic

The Statements of Propositional Logic

The statements of propositional logic are expressions who have a truth value which is either true or false.

**Definition 19.** Let \( P, Q \) be statements. Then the five expressions “\( \neg P \)”,” \( P \land Q \)”, “\( P \lor Q \)”, “\( P \Rightarrow Q \)”, and “\( P \Leftrightarrow Q \)” are also statements whose truth values are completely determined by the truth values of \( P \) and \( Q \) as shown in the following table:

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( \neg P )</th>
<th>( P \land Q )</th>
<th>( P \lor Q )</th>
<th>( P \Rightarrow Q )</th>
<th>( P \Leftrightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

We can also write ‘not’ for \( \neg \)’, ‘if and only if’ for \( \Leftrightarrow \)’, ‘implies’ for \( \Rightarrow \)’, ‘and’ for \( \land \)’, and ‘or’ for \( \lor \). A statement of the form \( P \Rightarrow Q \) is called a conditional statement or an implication, and can be written in English as ‘\( P \) implies \( Q \)’, ‘if \( P \) then \( Q \)’, ‘\( Q \) follows from \( P \)’, or ‘\( Q \), if \( P \)’.

**Definition 20.** The statements \( S \), of Propositional Logic consists of

1. Atomic Statements that do not contain any of the five logical operators, and
2. Compound Statements that are one of the five forms, \( \neg P \), \( P \land Q \), \( P \lor Q \), \( P \Rightarrow Q \), or \( P \Leftrightarrow Q \) where \( P \) and \( Q \) are any elements of \( S \).

**Note:** In compound statements we usually put parentheses around the statements \( P \) or \( Q \) involved. For instance if \( P \) is the statement ‘\( P \) or \( Q \)’ and \( Q \) is the statement ‘\( R \) and \( S \)’ then \( P \Rightarrow Q \) should be written

\[
(P \text{ or } Q) \Rightarrow (R \text{ and } S)
\]

in order to avoid the confusion that ‘\( P \) or \( Q \Rightarrow R \) and \( S \)’ might actually mean something like \( P \) or \( (Q \Rightarrow (R \text{ and } S)) \). In order to cut down on parentheses, we assign a precedence order for our operators, meaning we apply the operators in the following order (from highest to lowest):

<table>
<thead>
<tr>
<th>Precedence of Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentheses, brackets, ( () ), { }, [ ] etc.</td>
</tr>
<tr>
<td>not</td>
</tr>
<tr>
<td>and, or</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
</tr>
<tr>
<td>( \Leftrightarrow )</td>
</tr>
</tbody>
</table>

The Rules of Inference of Propositional Logic

We list these rules in recipe notation. A more succinct symbolic formulation of these rules is listed in the Playbook.
## Rules of Inference

<table>
<thead>
<tr>
<th>and +</th>
<th>and −</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show:</strong> $W$</td>
<td><strong>Show:</strong> $W$ and $V$</td>
</tr>
<tr>
<td><strong>Show:</strong> $V$</td>
<td><strong>Conclude:</strong> $W$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $W$ and $V$</td>
<td><strong>Conclude:</strong> $V$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$⇒+$</th>
<th>$⇒−$ (modus ponens)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume</strong> $W$</td>
<td><strong>Show:</strong> $W$</td>
</tr>
<tr>
<td><strong>Show:</strong> $V$</td>
<td><strong>Show:</strong> $W ⇒ V$</td>
</tr>
<tr>
<td>$←$</td>
<td><strong>Conclude:</strong> $V$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $W ⇒ V$</td>
<td><strong>Conclude:</strong> $V$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$⇔+$</th>
<th>$⇔−$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show:</strong> $W ⇒ V$</td>
<td><strong>Show:</strong> $W ⇔ V$</td>
</tr>
<tr>
<td><strong>Show:</strong> $V ⇒ W$</td>
<td><strong>Conclude:</strong> $W ⇒ V$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $W ⇔ V$</td>
<td><strong>Conclude:</strong> $V ⇒ W$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>or +</th>
<th>or − (proof by cases)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show:</strong> $W$</td>
<td><strong>Show:</strong> $W$ or $V$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $W$ or $V$</td>
<td><strong>Show:</strong> $W ⇒ U$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $V$ or $W$</td>
<td><strong>Show:</strong> $V ⇒ U$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $U$</td>
<td><strong>Conclude:</strong> $U$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>not + (proof by contradiction)</th>
<th>not − (proof by contradiction)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume</strong> $W$</td>
<td><strong>Assume</strong> not $W$</td>
</tr>
<tr>
<td><strong>Show:</strong> $→←$</td>
<td><strong>Show:</strong> $→←$</td>
</tr>
<tr>
<td>$←$</td>
<td>$←$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> not $W$</td>
<td><strong>Conclude:</strong> $W$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$→←+$</th>
<th>copy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show:</strong> $W$</td>
<td><strong>Show:</strong> $W$</td>
</tr>
<tr>
<td><strong>Show:</strong> not $W$</td>
<td><strong>Conclude:</strong> $W$</td>
</tr>
<tr>
<td><strong>Conclude:</strong> $→←$</td>
<td><strong>Conclude:</strong> $→←$</td>
</tr>
</tbody>
</table>

### Remarks:
- The symbol $←$ is an abbreviation for “end assumption”.
- The symbol $→←$ is called “contradiction” and represents the logical constant FALSE.
- The italicized word **Assume** is actually entered as part of the proof itself, it is not just an instruction in the recipe like the words 'Show:' and 'Conclude:'.
- The inputs “**Assume** -” and “$←$” are not themselves statements that you prove or are given, but rather are inputs to rules of inference that may be inserted into a proof at any time. There is no reason however, to insert such statements unless you intend to use one of the rules of inference that requires them as inputs.
- The statement following an **Assume** is the same as any other statement in the proof and can be used as an input to a rule of inference.
- Statements in an **Assume**$←$ block can be used as inputs to rules of inference whose conclusion is also inside the same block only. Once a **Assume** is closed with a matching $←$, only the entire block can be used as an input to a rule of inference. The individual statements within a block are no longer valid.
outside the block. We usually indent and Assume- block to keep track of what statements are valid under which assumptions.

**Definition 21.** A compound statement of propositional logic is called a *tautology* if it is true regardless of the truth values the atomic statements that comprise it. (Its "truth table" contains only T's.)

It can be shown that a statement can be proved with Propositional Logic if and only if the statement is a tautology.

### 6 Formal Proof vs. Traditional Proof Style

One way to write down a formal proof of a theorem is called a **two column proof**. This style of proof consists of a sequence of numbered lines containing statements, reasons, and references to premises. Every line contains exactly one statement (or declaration - see below), and the reason given on that line is the name of a rule of inference for which the statement on that line is the conclusion. If the rule of inference has premises, the reason is followed by the line numbers containing the statements (or variable declarations) which are the premises that the rule is being applied to. References to premises can only refer to lines which appear earlier in the same proof which are not contained in a subproof that has been closed. Subproofs used as a premise are cited by listing the range of line numbers comprising the subproof.

**Example 22.** Let $P$ and $Q$ be statements. Prove the following case of DeMorgan’s Law, namely that

$$\neg P \text{ or } \neg Q \Rightarrow \neg(P \text{ and } Q)$$

**Proof.**

1. Assume $\neg P$ or $\neg Q$  
2. Assume $\neg P$  
3. Assume $P$ and $Q$  
4. $P$ by and $\vdash; 3$  
5. $\rightarrow\leftarrow$ by $\rightarrow\leftarrow +; 2,4$  
6. $\leftarrow$  
7. $\neg(P$ and $Q)$ by not $+; 3,5,6$  
8. $\leftarrow$  
9. $\neg P \Rightarrow \neg(P$ and $Q)$ by $\Rightarrow +; 2,7,8$  
10. Assume $\neg Q$  
11. Assume $P$ and $Q$  
12. $Q$ by and $\vdash; 11$  
13. $\rightarrow\leftarrow$ by $\rightarrow\leftarrow +; 10,12$  
14. $\leftarrow$  
15. $\neg(P$ and $Q)$ by not $+; 11,13,14$  
16. $\leftarrow$  
17. $\neg Q \Rightarrow \neg(P$ and $Q)$ by $\Rightarrow +; 10,15,16$  
18. $\neg(P$ and $Q)$ by or $\vdash; 1,9,17$  
19. $\leftarrow$  
20. $\neg P$ or $\neg Q \Rightarrow \neg(P$ and $Q)$ by $\Rightarrow +; 1,18$  

Notice that when a rule of inference has a subproof for a premise, we indicate this by citing the line numbers for the assumption, the conclusion, and the end of assumption block indicator ($\leftarrow$) e.g., as shown in line 7 above.
A traditional proof would be more explanatory, like an essay, describing the steps and reasons in word-wrapped form. The above proof could be written in a traditional way as follows.

**Theorem 23.** Let $P$ and $Q$ be statements. Then $(\neg P \text{ or } \neg Q) \Rightarrow \neg(P \text{ and } Q)$.

**Proof.** We wish to show that if at least one of ‘not $P$’ or ‘not $Q$’ is true, then it is not the case that both $P$ and $Q$ are true.

So, assume $\neg P$ or $\neg Q$. We wish to show $\neg(P \text{ and } Q)$. As a shorthand, define $R$ to be the statement $\neg(P \text{ and } Q)$. We will prove $R$, using the “or $\neg$” rule of inference starting from $\neg P$ or $\neg Q$. In other words, we want to show that both $\neg P$ and $\neg Q$ imply $R$.

First, suppose $\neg P$ is true. Assume for contradiction that $P$ and $Q$. Then by the and $\neg$ rule we know $P$ is true, contradicting our assumption of $\neg P$. Thus by $\neg+$ (or proof by contradiction), we can conclude $\neg(P \text{ and } Q)$. Therefore:

$$\neg P \Rightarrow \neg(P \text{ and } Q).$$

Next, suppose $\neg Q$ is true. A similar argument to the previous paragraph shows that $\neg(P \text{ and } Q)$ is true in this case as well, so

$$\neg Q \Rightarrow \neg(P \text{ and } Q).$$

By the or $\neg$ rule, our proof is complete.

**Tips for Proof Writing:** There is not a single correct way to word wrap a formal proof into a traditional proof style, but we generally try to use the following guidelines to make our proofs more readable:

- Tell the reader what your proof strategy is up front. Notice in the proof above we say up front that we will be using the ‘or $\neg$’ rule of inference (which we could have also called “proof by cases”). This allows the reader to more easily follow the remainder of the argument.
- Never start a sentence with a math expression. We would never start the sentence with $\neg P$, for instance.
- Use “we” rather than “I”, as in “We now show that...” rather than “I will now show that...”
- Put important mathematical expressions in the proof in display mode - centered and on a line by themselves - in order to draw attention to them. You can also label such lines with a number on the right if you need to refer to them later.
- Skip the last few lines of the formal proof if they are clear to anyone familiar with the rules of inference.
- Skip details that will be clear to the reader, but don’t skip any important details! (In general, which details you can skip will depend somewhat on your audience.)
- Put an end-of-proof symbol, usually either $\Box$ or “QED”, to mark the end of your proof!

**Exercise 24.** Give a formal proof for the reverse case of DeMorgan’s Law, namely that

$$\neg(P \text{ and } Q) \Rightarrow \neg P \text{ or } \neg Q$$

**Exercise 25.** Give a formal proof for yet another case of DeMorgan’s Law, namely that

$$\neg(P \text{ or } Q) \Leftrightarrow \neg P \text{ and } \neg Q$$

7 Predicate Logic

We can extend Propositional Logic by adding more statements and rules of inference to those we already have in our formal system. This extended formal system is called **Predicate Logic**.
7.1 Quantifiers

The symbol \( \lambda \) in the lambda expression \((\lambda x, E)\) is an example of a quantifier. The thing that all quantifiers have in common is that they bind variables. If \( W \) is an expression that does not contain any quantifiers, then every occurrence of every identifier that appears in the expression is said to be a free occurrence of that identifier.

If a quantifier appears in an expression, there are one or more variables that it binds. All occurrences of the variables that are in the scope of the quantifier (usually everything to the right of it until a scope delimiter for that quantifier is encountered) are called bound variables.

Predicate logic extends propositional logic by defining two additional quantifiers.

Definition 26. The symbols \( \forall \) and \( \exists \) are quantifiers. The symbol \( \forall \) is called “for all”, “for every”, or “for each”. The symbol \( \exists \) is called “for some” or “there exists”.

We will encounter more quantifiers beyond just these two.

7.2 Statements

Every statement of Propositional Logic is still a statement of Predicate Logic. In addition we define the following statements.

Definition 27. If \( x \) is any variable and \( W \) is a lambda expression that simplifies to a statement when applied to any expression having the same type as \( x \), then \((\forall x, W(x))\) and \((\exists x, W(x))\) are both statements.

We say that the scope of the quantifier in \((\forall x, W(x))\) and \((\exists x, W(x))\) is everything inside the outer parentheses. Sometimes these parentheses are omitted when the scope is clear from context. All occurrences if \( x \) throughout the scope are said to be bound by the quantifier.

7.3 Variable declaration

Before using a free identifier for the first time in any expression in our proofs we should tell the reader what that identifier represents. There are four ways to introduce a new free identifier.

1. It can be declared to be a variable (a variable declaration).
2. It can be declared to be a constant (a constant declaration).
3. It can be defined as temporary new notation, usually as an abbreviation for a larger expression (a notational definition).
4. It can occur free in an expression preceding the proof itself, such as in the statement of the theorem, in a premise that is given, or declared globally prior to the start of the proof (globally declared).

Bound variables do not have to be declared. They can be any identifier you like, as long as that identifier is not in the scope of more than one quantifier that binds it.

7.4 Rules of Inference

The rules of inference for these two quantifiers are as follows. These are expressed in recipe notation.
Rules of Inference for Logical Quantifiers

<table>
<thead>
<tr>
<th>∀+</th>
<th>∀−</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let s be arbitrary (variable declaration)</td>
<td>Show: ∀x, W(x)</td>
</tr>
<tr>
<td>Show: W(s)</td>
<td>Conclude: W(t)</td>
</tr>
<tr>
<td>←</td>
<td>→←</td>
</tr>
<tr>
<td>Conclude: ∀x, W(x)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∃+</th>
<th>∃−</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show: W(t)</td>
<td>Show: ∃x, W(x)</td>
</tr>
<tr>
<td>Conclude: ∃x, W(x)</td>
<td>For some c (constant declaration)</td>
</tr>
<tr>
<td></td>
<td>Conclude: W(c)</td>
</tr>
</tbody>
</table>

*Restrictions and Remarks:

- In ∀+, s must be a new variable in the proof, cannot appear as a free variable in any assumption or premise, and W(s) cannot contain any constants which were produced by the ∃− rule. The indentation and ← symbol indicate the scope of the declaration of s. Variables s and x must have the same type.

- In ∀− and ∃+, no free variable in t may become bound when t is substituted for x in W(x). Variable x and expression t must have the same type.

- In ∃+, t can be an expression, and W(x) can be the expression obtained by replacing one or more of the occurrences of t with x. The identifier x cannot occur free in W(t). Variable x and expression t must have the same type.

- In ∃−, c must be a new identifier in the proof. Also W(c) must immediately follow the constant declaration for c in the proof. The scope of the declaration continues indefinitely or until the end of the scope of any subproof block or variable declaration scope that contains the constant declaration. Variable x and constant c must have the same type.

One consequence of this is that it enforces the restriction on ∀+ that prohibits any constant declared with ∃− to appear in W(s) because after the application of ∀+ any free occurrence of c is no longer in the scope of the original declaration (and therefore undeclared).

**Exercise 28.** *(Alpha Substitution.)* Show that the choice of bound variable doesn’t matter (for new variables in the proof) by giving a formal proof for

(∀x, P(x)) \Rightarrow (∀y, P(y))

and

(∃x, P(x)) \Rightarrow (∃y, P(y))

**Example 29.** Let’s prove yet another DeMorgan’s Law using a formal proof in our Predicate Logic.

\neg(∀x, P(x)) \iff (∃x, \neg P(x))

**Proof.**

1. Assume \neg(∀x, P(x)) -
2. Assume \neg(∃x, \neg P(x)) -
3. Let t be arbitrary -
4. Assume \neg P(t) -
5. ∃x, \neg P(x) ∃+; 4
6. \rightarrow← \rightarrow← +; 2, 5
7. ← -
8. $P(t)$  \(\neg\); 4, 7, 8
9. $\leftarrow$ $
eg$
10. $\forall x, P(x)$  $\forall$; 3, 8, 9
11. $\rightarrow\leftarrow$  $\rightarrow\leftarrow$; 1, 10
12. $\leftarrow$ $
eg$
13. $\exists x, \neg P(x)$  $\neg\neg$; 2, 11, 12
14. $\leftarrow$ $
eg$
15. $\neg (\forall x, P(x)) \Rightarrow (\exists x, \neg P(x))$  $\Rightarrow$; 1, 13, 14
16. Assume $\exists x, \neg P(x)$ $
eg$
17. For some $c$ const dec
18. $\neg P(c)$  $\exists$; 16, 17
19. Assume $\forall x, P(x)$ $
eg$
20. $P(c)$  $\forall$; 17, 19
21. $\rightarrow\leftarrow$  $\rightarrow\leftarrow$; 20, 18
22. $\leftarrow$ $
eg$
23. $\neg (\forall x, P(x))$  $\neg\neg$; 19, 22
24. $\leftarrow$ $
eg$
25. $(\exists x, \neg P(x)) \Rightarrow (\forall x, P(x))$  $\Rightarrow$; 16, 23, 24
26. $\neg (\forall x, P(x)) \Leftrightarrow (\exists x, \neg P(x))$  $\Leftrightarrow$; 15, 25

\[ \square \]

**Definition 30.** If $x, y$ are any variables of the same type and $W$ is a lambda expression not containing $x$ or $y$ that simplifies to a statement when applied to any expression having the same type as $x$ and $y$, we define

$$(\exists! x, W(x)) \Leftrightarrow \exists x, (W(x) \text{ and } \forall y, W(y) \Rightarrow y = x)$$

The statement $\exists! x, W(x)$ is read “There exists a unique $x$ such that $W(x)$.”

---

**Rules of Inference for Unique Existence**

<table>
<thead>
<tr>
<th>$\exists!+$</th>
<th>$\exists!-$</th>
</tr>
</thead>
</table>
| **Show:** $W(s)$ | **Show:** $\exists x, W(x)$  
**Let $y$ be arbitrary.** | **Conclude:** $\exists x, W(x)$ and $\forall y, W(y) \Rightarrow y = x$ |
| $\text{Assume } W(y)$ | |
| **Show:** $y = s$ | |
| $\leftarrow$ | |
| **Conclude:** $\exists! x, W(x)$ | |

---

**Equality**

**Definition 31.** The equality symbol, $=,$ is defined by the following two rules of inference.

---

**Rules of Inference for Equality**

<table>
<thead>
<tr>
<th>Reflexivity of $=$</th>
<th>Substitution*</th>
</tr>
</thead>
</table>
| **Conclude:** $x = x$ | **Show:** $x = y$  
**Show:** $W$  
**Conclude:** $W$ with the $n$th free occurrence of $x$ replaced by $y$. |

*Restrictions and Remarks*
• Note that in the Reflexive rule there are no inputs, so you can insert a statement of the form $x = x$
into your proof at any time.

• No free variable in $y$ can become bound when $y$ is substituted for $x$.

**Exercise 32.** Prove the following with a formal proof.

$$(\exists! x, P(x)) \land (P(a) \land P(b)) \Rightarrow a = b$$

**Precedence**

While this is not universally agreed on in mathematics, in our formal system, quantifiers have a lower
precedence than $\leftrightarrow$. Thus they quantify the largest statement to their right possible unless specifically
limited by parentheses. For instance,

$$\forall x, P(x) \Rightarrow Q$$

means the same thing as $\forall x, (P(x) \Rightarrow Q)$ but is different from $(\forall x, P(x)) \Rightarrow Q$. 