Chapter 2

Logic

This chapter covers propositional logic and predicate logic at a basic level. Some deeper issues will be covered later.

2.1 A bit about style

Writing mathematics requires two things. You need to get the logical flow of ideas correct. And you also need to express yourself in standard style, in a way that is easy for humans (not computers) to read. Mathematical style is best taught by example and is similar to what happens in English classes.

Mathematical writing uses a combination of equations and also parts that look superficially like English. Mathematical English is almost like normal English, but differs in some crucial ways. You are probably familiar with the fact that physicists use terms like "force" differently from everyone else. Or the fact that people from England think that "paraffin" is a liquid whereas that word refers to a solid substance in the US. We will try to highlight the places where mathematical English isn't like normal English.

You will also learn how to make the right choice between an equation and an equivalent piece of mathematical English. For example, $\wedge$ is a shorthand symbol for "and." The shorthand equations are used when we want to look at a complex structure all at once, e.g. discuss the logical structure of a proof. When writing the proof itself, it's usually better to use the longer English equivalents, because the result is easier to read. There is no hard-and-fast line here, but we'll help make sure you don't go too far in either direction.

2.2 Propositions 命题

Two systems of logic are commonly used in mathematics: propositional logic and predicate logic. We'll start by covering propositional logic.

A *proposition* is a statement which is true or false (but never both!). For example, "Urbana is in Illinois" or $2 \leq 15$. It can't be a question. It also can't contain variables, e.g. $x \leq 9$ isn't a proposition. Sentence fragments without verbs (e.g. "bright blue flowers") or arithmetic expressions (e.g. $5 + 17$), aren't propositions because they don't state a claim.

The lack of variables prevents propositional logic from being useful for very much, though it has some applications in circuit analysis, databases, and artificial intelligence. Predicate logic is an upgrade that adds variables. We will mostly be using predicate logic in this course. We just use propositional logic to get started.

2.3 Complex propositions

Statements can be joined together to make more complex statements. For example, "Urbana is in Illinois and Margaret was born in Wisconsin." To talk about complex sequences of statements without making everything too long, we represent each simple statement by a variable. E.g. if p is "Urbana is in Illinois" and q is "Margaret was born in Wisconsin", then the whole long statement would be "p and q". Or, using shorthand notation $p \wedge q$.

The statement $p \wedge q$ is true when both p and q are true. We can express this with a "truth table":

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Similarly, $\neg p$ is the shorthand for "not p." In our example, $\neg p$ would be "Urbana is not in Illinois." $\neg p$ is true exactly when p is false.

$p \vee q$ is the shorthand for "p or q", which is true when either p or q is true. Notice that it is also true when both p and q are true, i.e. its true table is:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

When mathematicians use "or", they always intend it to be understood with this "inclusive" reading.

Normal English sometimes follows the same convention, e.g. "you need to wear a green hat or a blue tie" allows you to do both. But normal English is different from mathematical English, in that "or" sometimes excludes the possibility that both statements are true. For example, "Clean up your room or you won't get desert" strongly suggests that if you do clean up your room, you will get desert. So normal English "or" sometimes matches mathematical "or" and sometimes another operation called **exclusive or** defined by

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Exclusive or has some important applications in computer science, especially in encoding strings of letters for security reasons. However, we won't see it much in this class.

2.4 Implication 推断

Two propositions p and q can also be joined into the **conditional** statement. "if p, then q." The proposition after the "if" (p in this case) is called the "hypothesis" and the proposition after "then" (q in this example) is called the "conclusion." As in normal English, there are a number of alternative ways to phrase the statement "if p, then q", e.g. "p implies q" or "q follows from p".

The shorthand for this conditional is $p \rightarrow q$ and its truth table is

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

For example, "If Obama is president, then Obama lives in the White House" is true (at least in 2010). But "If Obama is president, then $2 > 4$" is false. All the examples tend to be a bit artificial, because we don't have variables yet.

In normal English, we tend not to use conditionals in which the "if" part is false. E.g. "If Bush is president, then Urbana is in Illinois." In mathematical English, such statements occur more often. And, worse, they are always considered true, no matter whether the "then" part is true or false. For example, this statement is true: "If Bush is president, then $2 > 4$."

The easiest way to remember the right output values for this operation is to remember that the value is false in exactly one case: when p is true and q is false.

Normal English requires that conditional sentences have some sort of causal connection between the two propositions, i.e. one proposition is true because the other is true. E.g. "If Helen learns to write C++, she will get a good job." It would seem odd if we said "If Urbana is in Illinois, then Margaret was born in Wisconsin." because there's no reason why one follows from the other. In mathematical English, this statement is just fine: there doesn't have to be any causal connection.

In normal English if/then statements, there is frequently a flow of time involved. Unless we make a special effort to build a model of time, propositional logic is timeless. This makes the English motivating examples slightly awkward. It's not a big problem in mathematics, because mathematical proofs normally discuss a world that is static. It has a cast of characters (e.g. variables, sets, functions) with a fixed set of properties, and we are just reasoning about what those properties are. Only very occasionally do we talk about taking an object and modifying it.

In computer programming, we often see things that look like conditional statements, e.g. "if $x > 0$, then increment y". But these are commands for the computer to do something, changing its little world. whereas the similar-looking mathematical statements are timeless. Formalizing what it means for a computer program to "do what it's supposed to" requires modelling how the world changes over time. You'll see this in later CS classes.

2.5 Converse, contrapositive, biconditional

The converse of $p \rightarrow q$ is $q \rightarrow p$. The two statements are not equivalent. To see this, compare the previous truth table with this one:

p	q	$q \rightarrow p$
T	T	T
T	F	T
F	T	F
F	F	T

The converse mostly occurs in two contexts. First, getting the direction of implication backwards is a common bug in writing proofs. That is, using the converse rather than the original statement. This is a bug because implications frequently hold in only one direction.

Second, the phrase "p implies q, and conversely" means that p and q are true under exactly the same conditions. The shorthand for this is the biconditional operator $p \leftrightarrow q$.

p	q	$q \leftrightarrow p$
T	T	T
T	F	F
F	T	F
F	F	T

Another common way to phrase the biconditional is "p if and only if q."

The contrapositive of $p \rightarrow q$ is formed by swapping the roles of p and q and negating both of them to get $\neg q \rightarrow \neg p$. Unlike the converse, the contrapositive **is** equivalent to the original statement. Here's a truth table showing why:

p	q	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$
T	T	F	F	T
T	F	T	F	F
F	T	F	T	T
F	F	T	T	T

To figure out the last column of the table, recall that $\neg q \rightarrow \neg p$ will be false in only one case: when the hypothesis ($\neg q$) is true and the conclusion ($\neg p$) is false.

Let's consider what these variations look like in an English example:

- If it's below zero, my car won't start.
- converse: If my car won't start, it's below zero
- contrapositive: If my car will start, then it's not below zero.

2.6 Complex statements

Very complex statements can be made using combinations of connectives. E.g. "If it's below zero or my car does not have gas, then my car won't start and I can't go get groceries." This example has the form

$$(p \vee \neg q) \rightarrow (\neg r \wedge \neg s)$$

The shorthand notation is particularly useful for manipulating complicated statements, e.g. figuring out the negative of a statement.

When you try to read a complex set of propositions all glued together with connectives, there is sometimes a question about which parts to group together first. English is a bit vague about the rules. So, for example, in the previous example, you need to use common sense to figure out that "I can't go get groceries" is intended to be part of the conclusion of the if/then statement.

In mathematical shorthand, there are conventions about which parts to group together first. In particular, you apply the "not" operators first, then the "and" and "or". Then you take the results and do the implication operations. This is basically similar to the rules in (say) high-school algebra. Use parentheses if you intend the reader to group things differently, or if you aren't sure that your reader will group your statement as you intended.

You can build truth tables for complex statements, e.g.

p	q	r	$q \wedge r$	$(q \wedge r) \rightarrow p$
T	T	T	T	T
T	F	T	F	T
F	T	T	T	F
F	F	T	F	T
T	T	F	F	T
T	F	F	F	T
F	T	F	F	T
F	F	F	F	T

Truth tables are a nice way to show equivalence for compound propositions which use only 2-3 variables. However, if there are k variables, your table needs 2^k lines to cover all possible combinations of input truth values. This is cumbersome for larger numbers of variables.

2.7 Logical Equivalence

Two (simple or compound) propositions p and q are *logically equivalent* if they are true for exactly the same input values. The shorthand notation for this is $p \equiv q$. One way to establish logical equivalence is with a truth table.

For example, we saw that implication has the truth table:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

A frequently useful fact is that $p \rightarrow q$ is logically equivalent to $\neg p \vee q$.

To show this, we build the truth table for $\neg p \vee q$ and notice that the output values exactly match those for $p \rightarrow q$.

p	q	$\neg p$	$\neg p \vee q$
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

Two very well-known equivalences are *De Morgan's Laws.* These state that $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$. and that $\neg(p \vee q)$ is equivalent to $\neg p \wedge \neg q$. Similar rules in other domains (e.g. set theory) are also called De Morgan's Laws. They are especially helpful, because they tell you how to simplify the negation of a complex statement involving "and" and "or".

We can show this easily with another 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

T and F are special constant propositions with no variables that are, respectively, always true and always false. So, since $p \wedge \neg p$ is always false, we have the following equivalence:

$$p \wedge \neg p \equiv F$$

Notice that, in mathematics, the equal operator $=$ can only be applied to objects such as numbers. When comparing logical expressions that return true/false values, you must use $\equiv$. If use $\equiv$ to create complex logical equations, use indenting and whitespace to make sure the result is easy to read.

2.8 Some useful logical equivalences

It is easy to find (e.g. on the internet) long tables of useful logical equivalences. Most of them are commonsense and/or similar to rules from algebra. For example, $\wedge$ and $\vee$ are commutative, e.g. $p \wedge q \equiv q \wedge p$.

The distributive laws, however, work slightly differently from those in algebra. In algebra we have one rule:

$$a(b + c) = ab + ac$$

where as in logic we have two rules:

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

So, in logic, you can distribute either operator over the other. Also, arithmetic has a clear rule that multiplication is done first, so the righthand side doesn't require parentheses. The order of operations is less clear for the logic, so more parentheses are required.

2.9 Negating propositions

An important use of logical equivalences is to help you correctly state the negation of a complex proposition, i.e. what it means for the complex proposition not to be true. This is important when you are trying to prove a claim false or convert a statement to its contrapositive. Also, looking at the negation of a definition or claim is often helpful for understanding precisely what the definition or claim means. Having clear mechanical rules for negation is important when working with concepts that are new to you, when you have only limited intuitions about what is correct.

For example, suppose we have a claim like “If M is regular, then M is paracompact or M is not Lindelöf.” I'm sure you have no idea whether this is even true, because it comes from a math class you are almost certain not to have taken. However, you can figure out its negation.

First, let's convert the claim into shorthand so we can see its structure. Let r be "M is regular", p be "M is paracompact", and l be "M is Lindelöf." Then the claim would be $r \rightarrow (p \vee \neg l)$.

The negation of $r \rightarrow (p \vee \neg l)$ is $\neg(r \rightarrow (p \vee \neg l))$. However, to do anything useful with this negated expression, we normally need to manipulate it into an equivalent expression that has the negations on the individual propositions.

The key equivalences used in doing this manipulation are:

- $\neg(\neg p) \equiv p$
- $\neg(p \wedge q) \equiv \neg p \vee \neg q$
- $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- $\neg(p \rightarrow q) \equiv p \wedge \neg q$.

The last of these follows from an equivalence we saw above: $p \rightarrow q \equiv \neg p \vee q$ plus one of DeMorgan's laws.

So we have

$$\neg(r \rightarrow (p \vee \neg l)) \equiv r \wedge \neg(p \vee \neg l) \equiv r \wedge \neg p \wedge \neg\neg l \equiv r \wedge \neg p \wedge l$$

So the negation of our original claim is "M is regular and M is not paracompact and M is Lindelöf." Knowing the mechanical rules helps you handle situations where your logical intuitions aren't fully up to the task of just seeing instinctively what the negation should look like. Mechanical rules are also very helpful when you are tired or stressed (e.g. during an exam).

Notice that we've also derived a new logical equivalence $\neg(r \rightarrow (p \vee \neg l)) \equiv r \wedge \neg p \wedge l$ by applying a sequence of known equivalences. This is how we establish new equivalences when truth tables get unwieldy.

2.10 Predicates and Variables

Propositions are a helpful beginning but too rigid to represent most of the interesting bits of mathematics. To do this, we need predicate logic, which allows variables and predicates that take variables as input. We'll get started with predicate logic now, but delay covering some of the details until they become relevant to the proofs we're looking at.

A predicate is a statement that becomes true or false if you substitute in values for its variables. For example, "$x^2 \geq 10$" or "y is winter hardy." Suppose we call these $P(x)$ and $Q(y)$. Then $Q(y)$ is true if y is "mint" but not if y is "tomato".[1]

If we substitute concrete values for all the variables in a predicate, we're back to having a proposition. That wasn't much use, was it?

The main use of predicates is to make general statements about what happens when you substitute a variety of values for the variables. For example:

P(x) is true for every x

For example, "For every integer x, $x^2 \geq 10$" (false).

Consider "For all x, $2x \geq x$." Is this true or false? This depends on what values x can have. Is x any integer? In that case the claim is false. But if x is supposed to be a natural number, then the claim is true.

In order to decide whether a statement involving quantifiers is true, you need to know what types of values are allowed for each variable. Good style requires that you state the type of the variable explicitly when you introduce it, e.g. "For all natural numbers x, $2x \geq x$." Exceptions involve cases where the type is very, very clear from the context, e.g. when a whole long discussion is all about (say) the integers. If you aren't sure, a redundant type statement is a minor problem whereas a missing type statement is sometimes a big problem.

2.11 Other quantifiers

The general idea of a quantifier is that it expresses how many of the values in the domain make the claim true. Normal English has a wide range of

[1]A winter hardy plant is a plant that can survive the winter in a cold climate, e.g. here in central Illinois.

quantifiers which tell you roughly how many of the values work, e.g. "some", "a couple", "a few", "many", "most." For example, "most students in this class have taken a programming class."

By contrast, mathematics uses only three quantifiers, one of which is used rarely. We've seen the *universal quantifier* "for all." The other common one is the *existential quantifier* "there exists," as in

> There is an integer x such that $x^2 = 0$.

In normal English, when we say that there is an object with some properties, this tends to imply that there's only one or perhaps only a couple. If there were many objects with this property, we normally expect the speaker to say so. So it would seem odd to say

> There is an integer x such that $x^2 > 0$.

Or

> There exists an integer x such that $5 < x < 100$.

Mathematicians, however, are happy to say things like that. When they say "there exists an x," with certain properties, they mean that there exists at least one x with those properties. They are making no claims about how many such x's there are.

However, it is sometimes important to point out when one and only one x has some set of properties. The mathematical jargon for this uses the *unique existence* quantifier, as in:

> There is a unique integer x such that $x^2 = 0$.

Mathematicians use the adjective "unique" to mean that there's only one such object (similar to the normal usage but not quite the same).

2.12 Notation

The universal quantifier has the shorthand notation $\forall$. For example,

$$\forall x \in \mathbb{R}, x^2 + 3 \geq 0$$

In this sentence, $\forall$ is the quantifier. $x \in \mathbb{R}$ declares the variable and the set ($\mathbb{R}$) from which its values can be taken, called its *domain* or its *replacement set*. As computer scientists, we can also think of this as declaring the type of x, just as in a computer program. Finally, $x^2 + 3 \geq 0$ is the predicate.

The existential quantifier is written $\exists$, e.g. $\exists y \in \mathbb{R}, y = \sqrt{2}$. Notice that we don't write "such that" when the quantifier is in shorthand. The unique existence quantifier is written $\exists!$ as in $\exists! x \in \mathbb{R}, x^2 = 0$. When existential quantifiers are written in English, rather than shorthand, we need to add the phrase "such that" to make the English sound right, e.g.

There exists a real number y such that $y = \sqrt{2}$.

There's no deep reason for adding "such that." It's just a quirk about how mathematical English is written, which you should copy so that your written mathematics looks professional. "Such that" is sometimes abbreviated "s.t."

2.13 Useful notation

If you want to state a claim about two numbers, you can use two quantifiers as in:

$$\forall x \in \mathbb{R}, \forall y \in \mathbb{R}, x + y \geq x$$

This is usually abbreviated to

$$\forall x, y \in \mathbb{R}, x + y \geq x$$

This means "for all real numbers x and y, $x + y \geq x$" (which isn't true).

In such a claim, the two variables x and y might contain different values, but it's important to realize that they might also be equal. For example, the following sentence is true:

$$\exists x, y \in \mathbb{Z}, x - y = 0$$

We saw above that the statement "if p, then q" has the contrapositive "if $\neg q$, then $\neg p$." This transformation can be extended to statements with a quantifier (typically universal). For example, the statement

$$\forall x, \text{ if } p(x), \text{ then } q(x)$$

would have a contrapositive

$$\forall x, \text{ if } \neg q(x), \text{ then } \neg p(x)$$

Notice that the quantifier stays the same: we only transform the if/then statement inside it.

2.14 Notation for 2D points

When writing mathematics that involves 2D points and quantifiers, you have several notational options. You can write something like $\forall x, y \in \mathbb{Z}$ ("for any integers x and y") and then later refer to the pair (x, y). Or you can treat the pair (x, y) as a single variable, whose replacement set is all 2D points. For example, the following says that the real plane ($\mathbb{R}^2$) contains a point on the unit circle:

$$\exists (x, y) \in \mathbb{R}^2, x^2 + y^2 = 1$$

Another approach is to write something like

$$\exists p \in \mathbb{R}^2, p \text{ is on the unit circle}$$

When you later need to make precise what it means to be "on the unit circle," you will have to break up p into its two coordinates. At that point, you say that that since p is a point on the plane, it must have the form (x, y), where x and y are real numbers. This defines the component variables you need to expand the definition of "on the unit circle" into the equation $x^2 + y^2 = 1$.

2.15 Negating statements with quantifiers

Suppose we have a universal claim like $\forall x \in \mathbb{R}, x^2 \geq 0$. This claim will be false if there is at least one real number x such that $x^2 < 0$. In general, a statement of the form "for all x in A, $P(x)$" is false exactly when there is some value x in A such that $P(x)$ is false. In other words, when "there exists x in A such that $P(x)$ is not true". In shorthand notation:

$$\neg(\forall x, P(x)) \equiv \exists x, \neg P(x)$$

Similarly,

$$\neg(\exists x, P(x)) \equiv \forall x, \neg P(x)$$

So this is a bit like the de Morgan's laws: when you move the negation across the operator, you change it to the other similar operator.

We saw above how to move negation operators from the outside to the inside of expressions involving $\wedge$, $\vee$, and the other propositional operators. Together with these two new rules to handle quantifiers, we now have a mechanical procedure for working out the negation of any random statement in predicate logic.

So if we have something like

$$\forall x, P(x) \rightarrow (Q(x) \wedge R(x))$$

Its negation is

$$\begin{aligned}\neg(\forall x, P(x) \to (Q(x) \wedge R(x))) &\equiv \exists x, \neg(P(x) \to (Q(x) \wedge R(x)))\\ &\equiv \exists x, P(x) \wedge \neg(Q(x) \wedge R(x)))\\ &\equiv \exists x, P(x) \wedge (\neg Q(x) \vee \neg R(x))\end{aligned}$$

2.16 Binding and scope

A quantifier is said to "bind" the variable it defines. For example, in the statement

$$\forall x \in \mathbb{R}, x^2 + 3 \geq 0$$

the quantifier $\forall$ binds the variable x.

The "bound" variable in a quantification is only defined for a limited time, called the "scope" of the binding. This is usually the end of the quantified statement or the end of the line, but you sometimes have to use common sense about what the author intended. Parentheses are often used to indicate that the author intends a shorter scope.

If a variable hasn't been bound by a quantifier, or otherwise given a value or a set of replacement values, it is called "free." Statements containing free variables don't have a defined truth value, so they cannot be (for example) a step in a proof.

Variables bound by quantifiers are like the dummy variables used in summations and integrals. For example, the i in $\sum_{i=0}^{n} \frac{1}{i}$ is only defined while you are still inside the summation. Variables in computer programs also have a "scope" over which their declaration is valid, e.g. the entire program, a single code file, or local to an individual function/procedure/method. If you try to use a variable outside the scope of its definition, you'll get a compiler error.

When writing mathematics, variables have to be defined, just like variables in computer programs. It's not polite to start using a variable without telling the reader where this variable comes from: is it bound by a quantifier? is it being set to a specific value (e.g. let $x = 3.1415$)? The reader also needs to know what type of value this variable might contain.

2.17 Variations in Notation

Although the core concepts of predicate logic are very standard, a few details vary from author to author. Please stick to the conventions used above, because it's less confusing if we all use the same notation. However, don't be surprised if another class or book does things differently. For example:

- There are several conventions about inserting commas after the quantifier and/or parentheses around the following predicate. We won't be picky about this.
- Some subfields (but not this class) have a convention that "and" is applied before "or," so that parentheses around "and" operations can be omitted. We'll keep the parentheses in such cases.
- Some authors use certain variations of "or" (e.g. "either ... or") with an exclusive meaning, when writing mathematical English. In this class, always read "or" as inclusive unless there is a clear explicit indiciation that it's meant to be exclusive. For example, "Take some bread or some cereal" should be read as inclusive in a mathematical context. If we wanted to indicate an exclusive reading, we would write something like "Take bread or some cereal, but not both."
- In circuits and certain programming languages, "true" and "false" are represented by 1 and 0. There are a number of different shorthand notations for logical connectives such as "and". Finally, programmers often use the term "boolean" to refer to true/false values and to expressions that return true/false values.