Propositional Logic
Propositions

A **proposition** is a declaration of fact that is either true or false, but not both. Examples and non-examples:

- **One plus two equals four** (proposition)
- **Mozart is the greatest composer** (not a proposition)
- **The equation 2x=1 has one real solution** (proposition)
- **2x=1** (not a proposition)
- **a^2 + b^2 = c^2** (not a proposition)
- **E = mc^2** (not a proposition)

Frequently, we will use the word “statement” instead of “proposition”.

Propositional variables and truth values

To represent propositions, we use variables, just like we represent quantities with variables. Common variable names for propositions are \( p, q \) and \( r \). We use the letters \( T \) and \( F \) to represent the truth values true and false.

When you assign a propositional variable to an equation, make certain to use proper parentheses. **Distinguish clearly between a value, and the statement that two quantities have that same value.**

Example:

\[ p = (1 + 2 = 3) \] defines \( p \) to be the statement that \( 1 + 2 \) equals \( 3 \). This \( p \) is a proposition.

\[ p = 1 + 2 = 3 \] defines \( p \) to be the number \( 3 \). This \( p \) is a quantity, not the statement that \( 1 + 2 \) and \( 3 \) are equal.

This distinction will be important later when we learn inductive proofs.
Compound propositions and logical operators

**Logical operators** aka logical connectives are symbols that create new propositions, called compound propositions, from existing propositions.

A logical operator is defined by its behavior, i.e. by which truth value the output proposition has for each configuration of truth values of the input propositions. A complete list of this information is called a truth table.
## Common Logical Operators

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Commonly translated as</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>not</td>
<td>¬</td>
</tr>
<tr>
<td>Conjunction</td>
<td>and</td>
<td>∧</td>
</tr>
<tr>
<td>Disjunction</td>
<td>(inclusive) or</td>
<td>∨</td>
</tr>
<tr>
<td>Exclusive Disjunction</td>
<td>exclusive or, xor</td>
<td>⊕</td>
</tr>
<tr>
<td>Conditional</td>
<td>If .. then, ..implies..</td>
<td>→</td>
</tr>
<tr>
<td>Biconditional</td>
<td>if and only if, iff</td>
<td>↔</td>
</tr>
</tbody>
</table>
Negation

The negation of p is the statement “it is not true that p”, also written as \( \neg p \). It is true when p is false, and false when p is true. Equivalently, negation is defined by the following truth table:

<table>
<thead>
<tr>
<th>p</th>
<th>( \neg p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

There are other symbols in use for negation, such as the tilde (\( \sim \)), the minus sign (\( \neg \)) and the exclamation point (!). The ! is very common in programming languages, as is “not”.

When you negate a statement, do not yield to the temptation to formally (and meaninglessly) apply the negation symbol to objects inside the statement. Objects cannot be negated, only statements. Incorrect negation:

\[
\neg(\text{Jack is taller than Joe}) \equiv (\neg\text{Jack is taller than } \neg\text{Joe})
\]

There are no such persons as “not Jack” or “not Joe”. You cannot negate a person. The correct negation is:

\[
\neg(\text{Jack is taller than Joe}) \equiv \text{Jack is at most as tall as Joe}.
\]
Negation Rules for Inequalities

The last slide contained an example of a subtlety you may have missed: the negation of a strict inequality is non-strict inequality and vice versa.

\[ \neg(x > y) \equiv (x \leq y) \]
\[ \neg(x \geq y) \equiv (x < y) \]

Therefore, if it is false that Jane is richer than Bill, then you cannot conclude that Bill is richer than Jane. The correct conclusion is that Bill is at least as rich as Jane.

You must exercise special care when negating a double inequality.

Let us consider the inequality \( 1 \leq x \leq 2 \). If you blindly apply the rule above, you will find the negation of the inequality as \( 1 > x > 2 \), or \( 2 < x < 1 \). But this makes no sense— if 2 is less than x, and x is less than 1, then 2 must be less than 1, a contradiction.

(The correct negation of \( 1 \leq x \leq 2 \) is

\[ x < 1 \lor x > 2 \]

We will come back to that in the next presentation.)
Conjunction

The conjunction of \( p \) and \( q \) is the statement “\( p \) and \( q \)”, also written as \( p \land q \). It is true when both \( p \) and \( q \) are true, otherwise, it is false.

Equivalently, conjunction is defined by the following truth table:

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( p \land q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Conjunction is usually represented by \&, && or “and” in programming languages.
Expressing Conjunction in English

In English, we often express the conjunction of statements that have the same subject and verb, but different objects, by stating subject and verb only once, and then connecting the objects by the conjunction. Instead of saying

*Mike buys a gallon of milk and Mike buys a loaf of bread,*

we say

*Mike buys a gallon of milk and a loaf of bread.*

The latter is formally incorrect because *and* connects only *statements*, and “a gallon of milk” and “a loaf of bread” are not statements.

Likewise, when two statements only share the subject, but have different verbs, we usually mention the subject only once and connect the incomplete (subject-less) sentences by conjunction. Instead of saying

*Mike buys a plane ticket and Mike rents a car,*

we say

*Mike buys a plane ticket and rents a car.*
Disjunction

The disjunction of p and q is the statement “p or q”, also written as \( p \lor q \). It is true when at least one of the statements p, q is true, otherwise, it is false. Disjunction is always inclusive by default.

Equivalently, disjunction is defined by the following truth table:

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p \lor q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Disjunction is usually represented by |, || or “or” in programming languages.
Exclusive Disjunction (aka Exclusive Or)

The exclusive disjunction of \( p \) and \( q \) is the statement “either \( p \) or \( q \) (but not both)”, also written as \( p \oplus q \). It is true when exactly one of the two statements \( p \), \( q \) is true, otherwise, false.

Equivalently, exclusive disjunction is defined by the following truth table:

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( p \oplus q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
The conditional $p \rightarrow q$ is the statement “if $p$, then $q$”. It is true exactly under the following conditions: when $p$ and $q$ are both true, or when $p$ is false.

In a conditional $p \rightarrow q$, $p$ is also known as the **premise**, and $q$ is known as the **conclusion**.

Equivalently, the conditional is defined by the following truth table:

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \rightarrow q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

**Do not confuse** the logical conditional with the *if-then* control structure that most programming languages have. *If (condition) then (code)* checks whether condition is true, and executes code in exactly that case. That is not the same thing as a logical conditional operator (not typically available as a built-in operator in programming languages) that would take two logical expressions $p, q$ and compute the truth value of $p \rightarrow q$. 
Conditionals in English

We will see later that there are many equivalent ways of expression conditionals in English. Some of them are just rearrangements of the standard *if*. *then* phrasing. Some omit the word “then”. For example, all of the following mean the same:

*If the sun shines, then we will go to the beach.*
*If the sun shines, we will go to the beach.*
*We will go to the beach if the sun shines.*

You must abstain from simple position-based reasoning when you parse conditionals. It is **not true** that whatever comes first is the premise, and what comes second is the conclusion.
Making sense of the truth table of the conditional

The conditional $p \rightarrow q$ is like a promise. It states that when $p$ is true, then $q$ must also be true. But it is not saying anything about the case when $p$ is false. If $p$ is false, the promise was never put to the test, so we have no grounds for holding it to be broken.

Example: A candidate running for office promises “If I get elected, I will create more jobs”. The candidate then fails to be elected. Whether or not jobs are subsequently created has no relevance to her campaign promise. Even if jobs are not created, we have no right to say that she broke her promise.

It would not be unreasonable to argue that if $p$ is false, the truth value of the conditional $p \rightarrow q$ should be “neutral” or perhaps “undefined”. However, we are doing binary logic here – by definition, a statement is either true or false. There is no neutral or undefined. Therefore, since we have no grounds to call $p \rightarrow q$ false when $p$ is false, we have no choice but to call it true. The candidate gets the benefit of the doubt.
The biconditional \( p \leftrightarrow q \) is the statement “if p, then q, and only then”. Informally, it is the “conditional going both ways”. A possible formal definition is

\[
p \leftrightarrow q \equiv (p \rightarrow q) \land (q \rightarrow p)
\]

The truth table of the biconditional is the following:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Observe that \( p \leftrightarrow q \) is the negation of \( p \oplus q \). Also observe that the biconditional is true exactly when \( p \) and \( q \) have the same truth value, whether that is true or false.
“Necessary” and “Sufficient” type statements

Conditional relationships are often expressed using the words “necessary” and “sufficient”.

Example: rain is necessary for a rainbow. This is another way of saying if there is a rainbow, there must be rain.

Example: money is sufficient for happiness. This is saying that if you have money, then you will be happy.

These examples illustrate the general rule:

“p is necessary for q” means $q \rightarrow p$

“p is sufficient for q” means $p \rightarrow q$

“p is necessary and sufficient for q” means $p \leftrightarrow q$
Study Advice for “sufficient” type statements (1)

It is not a very good idea to attempt to reduce the understanding of “necessary” and “sufficient” type statements to the mindlessly symbolical level. For example, you may be tempted to memorize that whatever statement is to the left of “sufficient” becomes the $p$, the statement to the right becomes the $q$, and the whole statement is then $p \rightarrow q$.

The flexibility of the English language will foil this nice plan of yours. Let us study the word orders of different ways in which the conditional “if you win the lottery, then you will be rich” can be expressed as a “sufficient” type phrase.

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>A lottery win is sufficient for becoming rich.</td>
<td>$pSq$</td>
</tr>
<tr>
<td>To become rich, a lottery win is sufficient.</td>
<td>$qpS$</td>
</tr>
<tr>
<td>A sufficient condition to become rich is a lottery win.</td>
<td>$Sq$p</td>
</tr>
<tr>
<td>To become rich, it suffices to win the lottery.</td>
<td>$qSp$</td>
</tr>
</tbody>
</table>

In this table, $S$ signifies “is sufficient” or words to that effect, $p$ represents words to the effect of winning the lottery, and $q$ represents words to the effect of becoming rich.

As you can see, word order rules such as the “left-right” rule quoted above do not work here. You have to understand what the English sentence is saying.
Since word order rules do not work, let us try to understand “sufficient” type statements on the verbal level, through a synonym. To say that something “suffices” for something else is to say that it guarantees it.

“Eating apple pie is sufficient for eating apples” means that if you eat apple pie, you are guaranteed to eat apples. Therefore, the sentence is expressing the conditional eating apple pie → eating apples.

Another way of understanding “sufficient” type statements is to realize that whatever is “sufficient” in such a statement is the premise of the conditional.
Study Advice for “necessary” type statements (1)

Just like “sufficient” type phrases in English defy simplistic word order analysis, so do “necessary” type statements.

Let us study the word orders of different ways in which the conditional “if you can see the stars, then the sky is cloudless” can be expressed as a “necessary” type phrase.

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cloudless sky is necessary to see the stars.</td>
<td>$pNq$</td>
</tr>
<tr>
<td>To see the stars, a cloudless sky is necessary.</td>
<td>$qpN$</td>
</tr>
<tr>
<td>It is necessary for seeing the stars to have a cloudless sky.</td>
<td>$Nqp$</td>
</tr>
<tr>
<td>To see the stars, it is necessary to have a cloudless sky.</td>
<td>$qNp$</td>
</tr>
</tbody>
</table>

In this table, N signifies “is necessary” or words to that effect, p represents words describing the condition of a cloudless sky, and q represents words that express the ability to see the stars.
Study Advice for “necessary” type statements (2)

Again, we should try to understand “necessary” type statements on the verbal level. It is helpful to change “necessary” to “precondition”. For example, we rephrase

“A cloudless sky is necessary to see the stars” as
“A cloudless sky is a precondition for seeing the stars.”

The word precondition makes it clear that the cloudless sky does not guarantee that we can see the stars (it could be daytime). It is just one of several requirements. What we do know is this: if we can see the stars, then we must have cloudless sky.

Another way of understanding “necessary” type statements is to realize that whatever is “necessary” in such a statement is the conclusion of the conditional.
The “only if” statement

Sometimes, conditionals are phrased using the “only if” clause. We will again start with an example first.

A father tells his son: “you will get $100 only if you get good grades.”

The father is not saying that good grades will automatically lead to the disbursement of $100. He is only saying that good grades are a necessary precondition. Thus, if the son earns good grades, he may get the $100, but he may well have to satisfy other conditions.

All we know for certain is that if the son earned the $100, then he must have had good grades. Therefore, the general rule is:

“p only if q” means $p \rightarrow q$

If you find this hard to remember, just translate the conditional as if the word only was not there, and then switch premise and conclusion.

A consequence of this understanding is that the biconditional $p \leftrightarrow q$ can (and frequently is) expressed as “p if and only if q”, or “p iff q” for short.
Study Advice for “only if” type statements

Just like the other types of phrasings of conditionals we have discussed, “only if” statements defy simple positional analysis. Let an example make this clear:

*I’ll go to the beach tomorrow only if the sun shines.*

carries the same meaning as

*Only if the sun shines will I go to the beach tomorrow.*

The advice given on the previous slide bears repeating: if you are having difficulty parsing “only if” statements, eliminate the word “only”, and then switch premise and conclusion. Therefore, the only if statement above means:

*If I’m going to the beach tomorrow, then the sun must be shining.*
The “unless” statement

Example: a doctor tells her patient, “you will get sick unless you take the medicine”.

This doctor is not promising that the patient will be healthy if the medicine is taken. The patient could still get sick even on the medicine. She only promises that NOT taking the medicine will guarantee getting sick. Thus, what the doctor is saying is: “If you don’t take your medicine, you will get sick.”

This illustrates the following general translation rule:

“p unless q” means \( \neg q \rightarrow p \)

This rule shows that unless means if not. “You will get sick unless you take the medicine” means “You will get sick if you don’t take the medicine.”

You must again not jump to conclusions based on word order. The two components of the unless sentence can be given in either order:

Unless you take the medicine, you will get sick means the same as You will get sick unless you take the medicine.
Expressing a Conditional in Different Ways: Example 1

Let us express the conditional “if you can see the stars, then the sky must be clear” in different equivalent ways:

➢ You can see the stars *only if* the sky is clear.
➢ You can’t see the stars *unless* the sky is clear.
➢ You can’t see the stars *if* the sky is *not* clear.
➢ Clear sky is a *necessary* condition for seeing the stars.
➢ Seeing the stars is a *sufficient* condition for clear sky.
Expressing a Conditional in Different Ways: Example 2

Problem: select the statement that is logically equivalent to “If you can dream it, then you can achieve it.”

1. You can’t achieve it unless you dream it.
2. You can achieve it only if you dream it.
3. Dreaming it is a necessary condition for achieving it.
4. Achieving it is a necessary condition for dreaming it.
5. Achieving it is a sufficient condition for dreaming it.

Let us introduce the following variables:

\[ p = \text{“You can dream it”} \quad q = \text{“You can achieve it”}. \]

Then the original conditional is the statement \( p \rightarrow q \). Only the 4th of the statements above is logically equivalent to that. All the others are equivalent to \( q \rightarrow p \).
Variations of Conditional Statements

Given a conditional $p \rightarrow q$, there are several named modifications of this statement:

$q \rightarrow p$ is called the **converse** of $p \rightarrow q$.
$\neg p \rightarrow \neg q$ is called the **inverse** of $p \rightarrow q$.
$\neg q \rightarrow \neg p$ is called the **contrapositive** of $p \rightarrow q$.

Example:
original conditional C: *if it is snowing, we can build a snowman.*
converse of C: *if we can build a snowman, then it is snowing.*
inverse of C: *if it is not snowing, we can’t build a snowman.*
contrapositive of C: *if we can’t build a snowman, then it is not snowing.*

We will learn in a subsequent presentation that these different conditionals are **not** all logically equivalent.
Compound statements that involve two or more different logical operators need an *order of operation* (also known as operator precedence) in order to have unambiguous meaning. The following precedence is conventionally used, in order from highest to lowest:

1. negation 2. conjunction 3. disjunction 4. conditional 5. biconditional

Examples:

\[ p \lor q \land r \text{ means } p \lor (q \land r) \text{ because conjunction has a higher precedence than disjunction.} \]

\[ p \rightarrow q \lor r \text{ means } p \rightarrow (q \lor r). \]

We can use parentheses to override the operator precedence.
Left to Right or Right to Left?

When you have a chain of more than two conjunctions, or more than two disjunctions, the question comes up whether the evaluation is from right to left, or from left to right. Does \( a \lor b \lor c \) mean \((a \lor b) \lor c\) or \(a \lor (b \lor c)\)?

Fortunately, as we will learn in the next lecture, conjunction and disjunction are *associative*, which means that \((a \lor b) \lor c\) and \(a \lor (b \lor c)\) always have the same truth value, and the same goes for the expressions \((a \land b) \land c\) and \(a \land (b \land c)\).

We have no such luck with the conditional, which is not associative. This means that \((a \rightarrow b) \rightarrow c\) and \(a \rightarrow (b \rightarrow c)\) may have different truth values. There is no universal convention for whether the conditional is *left-associative* or *right-associative*. This makes expressions like \(a \rightarrow b \rightarrow c\) ambiguous in the absence of a local convention.
Logical Operators on Bitstrings

A *bitstring* is a sequence of binary digits (bits), i.e. 0’s and 1’s. For example, 01001 is a bitstring of length 5. The bitstring 1001 is a different bitstring than 01001.

We can extend the binary logical operators to apply to bitstrings of equal length, by interpreting 0 as false and 1 as true, and applying the logical operators to corresponding bits separately. We will write the bitstring logical operators with uppercase letters to distinguish them from the elementary logical operators. Examples:

<table>
<thead>
<tr>
<th>bitwise and</th>
<th>bitwise or</th>
<th>bitwise xor</th>
<th>bitwise →</th>
<th>Bitwise ↔</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000</td>
<td>101000</td>
<td>101000</td>
<td>101000</td>
<td>101000</td>
</tr>
<tr>
<td>AND 011011</td>
<td>OR 011011</td>
<td>XOR 011011</td>
<td>IMPL 011011</td>
<td>IFF 011011</td>
</tr>
<tr>
<td>= 001000</td>
<td>= 111011</td>
<td>= 110011</td>
<td>= 011111</td>
<td>= 001100</td>
</tr>
</tbody>
</table>

*Bitwise not* acts on a single bitstring. Example: NOT 101000 = 010111.