Introduction

What is logic?

Etymology Dictionary:

> *Logic*, from Greek
>
> *logos*: "reason", "idea", "word",
> *logike*: "(the) reasoning (art)".

Oxford English Dictionary:

> *Logic*, Science of "the forms of thinking in general, and more especially of inference and of scientific method. Also [...] symbolic techniques and mathematical methods to establish truth-values in the physical sciences, in language, and in philosophical argument" (OED).

What is logic? ctd.

Logic is the "calculus of computing": a mathematical foundation for dealing with information, and reasoning about program behaviour.

In this course, we are concerned with using logic to *describe*, *specify*, *verify*, and *reason* about *software*.

(We are not concerned with low-level "logic gates" etc in computing — that is for the Hardware course.)

Multiple different species of logic. E.g., logics for:

– Claims that can be true or false.

– Belief and knowledge.

– Ethics and law.

– Computer systems and how they behave over time.

What is logic? ctd.

A logic usually consists of:

- *Syntax* — a formal language (like a programming language) used to express concepts.
- *Semantics* — provides meaning for the formal language.
- *Proof theory* — a purely syntactic way of obtaining or identifying the valid statements of the language. It provides a way of arguing in the formal language.

We will study two specific logics:

- Propositional logic.
- First-order logic (FOL), also called 'predicate calculus' and 'predicate logic'.

These are foundational for any further logics you might later use. The former can be seen as a less fine-grained version of the latter.

Why learn logic?

- It provides good training in correct reasoning, and accurate, unambiguous description.
- Some of the more sophisticated real-world application areas need and use logic. For example:
  - *Answer set programming* is a logic programming paradigm used for building decision support systems, reconfiguring railway safety, etc.
  - The *SQL database language*, *model checking* and *controller synthesis* are based on logic.
  - Logic can shed light on “forms of thinking" — relevant to human-centric, explainable AI, safe AI and AGI.
    * *Logic-based machine learning* enables learning logical theories that support scientific knowledge discover.
    * *Planning and scheduling* uses logic to formalise a task and compute solutions for it.

Relevance to other modules

First year	Rest of this course	Intro to Databases	…		
second year	Models of Computation	Symbolic Reasoning	Intro to Prolog	…	
Third year	Theory & Practice of Concurrent Programming	Logic-based Learning	Advanced Databases	Distributted Algorithms	…
Fourth year	Knolwedge Representattion	Scalable Software Verification	Modal Logic and Strategic Reasoning for AI	Software Reliability	…

1. Propositional Logic

Main features

Consider the following example (the study of "if-tests")

```
if count>0 and not found then
   decrement count; look for next entry;
end if
```

- basic sentences (*propositional atoms*) — here: '`count>0`', `found` — are true or false depending on circumstances.
- connectives — `and, or, not,` etc. — used to build more and more complex test sentences from the atoms.
- the final complex sentence evaluates to true or false.

Propositional logic is not very expressive. FOL (later) can say much more, e.g., "every student has a tutor".

Why do propositional logic?

– All logics are based on propositional logic in some form.

– In programming, we want to handle arbitrarily complicated "if-tests" and study their general features.

  * *Evaluate* complicated "if-tests".

  * *Find out* when two "if-test" mean the same, when one implies another, whether an "if-test" (or "loop-test") can ever be made true or not.

– Propositional logic encapsulates an important class of computational problems, and so comes in to algorithms, complexity theory and SAT solving.

1.1 Syntax — The formal language

First, we need to fix a precise definition of the formal language, that is the *syntax* of propositional logic.

It has *three ingredients*:

1. Propositional atoms;
2. Boolean connectives;
3. Punctuation.

1st Ingredient: Propositional atoms

We are not concerned about which facts are being represented (‘`count>0`’, `found`). So long as they can get a *truth-value* (true or false), that’s enough. (Quotation marks are used to encapsulate a single expression.)

So we fix a collection of algebraic symbols to stand for these statements.

These symbols are called *propositional atoms.* Many people call them *propositional variables* or *propositional letters* instead. For short, we will usually call them *atoms.*

They are like variables $x, y, z, ...$ in maths. But because they are Propositional, we usually use the letters p, p', p'', $\ldots$ p_0, p_1, p_2, $\ldots$, and also p, q, r, s, $\ldots$. (Avoid mixing conventions.)

2nd Ingredient: Boolean connectives

We are interested in the following Boolean connectives (a.k.a. operator or operations):

- `not`: written as $\neg$ (or sometimes $\sim$ or $-$)
- `and`: written as $\wedge$ (or sometimes &, and in old books, '$\cdot$')
- `or`: written as $\vee$ (in old books, $+$ or v)
- `if-then`, or `implies`. This is written as $\rightarrow$ (or sometimes $\supset$, but not as $\Rightarrow$, which is used differently)
- `if-and-only-if`: written $\leftrightarrow$ (but not as $\Leftrightarrow$ or $\equiv$, which are used differently)
- `truth` and `falsity`: written as $\top$, $\bot$

We'll discuss the *meaning* of these connectives later.

Examples

Our test `count>0 and not found` would be expressed as `'count>0'` $\wedge$ $\neg$`found`, or better $p \wedge \neg q$.

Boolean connectives `and`, `or`, `implies`, `if-and-only-if` take two arguments and are written in infix form:
$p \wedge q$, $p \vee q$, $p \rightarrow q$, $p \leftrightarrow q$.

Negation (`not`) takes one argument, written to the right of it: e.g., $\neg p$, $\neg q$.

Truth and falsity, $\top, \bot$, take no arguments.
They are logical *constants* (like π, e in maths).

You will have to learn these new symbols, so you can read logic books. They are also quicker to write than `and, or, not`, etc.

3rd Ingredient: Punctuation

Consider the following *if-test.*

`‘count>0’` $\wedge$ $\neg$`found` $\vee$ `‘count>10’`

We need brackets to disambiguate it.

This is like terms in arithmetic: $1 - 2 + 3$ is ambiguous, we can read it as $(1 - 2) + 3$ or $1 - (2 + 3)$, and the difference matters.

For example, $p \wedge q \vee r$ might be read as

- $(p \wedge q) \vee r$,
- $p \wedge (q \vee r)$,

and the difference matters.

So we start by putting all brackets in, and then deleting those that are not needed.

Propositional formulas

What we have called *if-tests* are called *formulas* by logicians. (Some call them *sentences*, others call them *well-formed formulas,* or *wffs* for short.)

Informally, a *propositional formula* is a string of symbols made from propositional atoms, Boolean connectives, and brackets, in the appropriate way.

Definition 1.1 (Propositional formula)

- Any propositional atom (p, q, r, etc) is a propositional formula.
- $\top$ and $\bot$ are formulas.
- If ϕ is a formula then so is $(\neg\phi)$. *Note the brackets!*
- If ϕ, ψ are formulas then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \to \psi)$, and $(\phi \leftrightarrow \psi)$.
- That's it: nothing is a formula unless built by these rules.

The first two in the above are called *atomic* formulas.

Greek letters (ϕ, ψ, ...) are often used as meta-variables, representing formulas. They are not themselves expressions of the language.

Examples of formulas

The following are formulas:

- p
- $(\neg p)$
- $((\neg p) \wedge \top)$
- $(\neg((\neg s) \wedge \bot))$
- $((\neg p) \to (\neg((\neg q \vee r) \wedge \top)))$

How about these?

- $\wedge\ p\ q$ NO ($\wedge$ takes two arguments on either side.)
- $(\neg\bot) \wedge (r \to q)$ NO (missing brackets)
- $(p\ \vee q\ \neg s)$ NO
- $\neg r)$ NO

Examples of formulas ctd.

Take the formula $((\neg p) \to (\neg((\neg q \vee r) \wedge \top)))$.

You can see the brackets are a pain already. We need some conventions to get rid of (some of) them.

What we'll get are (strictly speaking) *abbreviations* of genuine formulas. But most people seem to think of them as real formulas.

We can always omit the final, outermost brackets:
$\neg p$, and $(\neg p) \to (\neg((\neg p) \wedge \top))$ are unambiguous.

Binding conventions

To get rid of more brackets, we *order* the Boolean connectives according to decreasing binding strength:

(*strongest*) $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$ (*weakest*)

This is like in arithmetic, where $\times$ is stronger than $+$. This means that $2+3\times 4$ is usually read as $2+(3\times 4)$, not as $(2+3)\times 4$. So:

- $p \vee q \wedge r$ is read as $p \vee (q \wedge r)$, not as $(p \vee q) \wedge r$.
- $\neg p \wedge q$ is read as $(\neg p) \wedge q$, not as $\neg(p \wedge q)$.

How about the following?

- $p \wedge \neg q \to r$ is read as $(p \wedge (\neg q)) \to r$, rather than $p \wedge (\neg(q \to r))$ or $p \wedge ((\neg q) \to r)$.

But don't take the conventions too far, e.g, $p \to \neg q \vee \neg\neg r \wedge s \leftrightarrow t$ is a mess! USE BRACKETS if it helps readability, even if they are not strictly needed.

Repeated connectives

What about $p \to q \to r$? The binding conventions are no help now. Should we read it as $p \to (q \to r)$ or as $(p \to q) \to r$?

There probably is a convention here. But I would always put brackets in such a formula. However, $p \wedge q \wedge r$ and $p \vee q \vee r$ are OK as they are.

As we will see, their *logical meaning* is the same, however we bracket them. $(p \wedge q) \wedge r$ and $p \wedge (q \wedge r)$ are *different formulas*. But we will see that they always have the same truth value in any situation: they are *logically equivalent*.

So we don't really care how we disambiguate $p \wedge q \wedge r$. (Usually, it is $(p \wedge q) \wedge r$ — from left, and we say in this case the connectives are left-associative.)

Repeated negations, as with $(\neg(\neg(\neg p)))$, can create no ambiguity: their brackets can always be removed, e.g., to $\neg\neg\neg p$.

Special cases

Sometimes the binding conventions don't work as we would like.

For example, if I saw

$$p \to r \wedge q \to r,$$

I'd probably read it as $(p \to r) \wedge (q \to r)$.

How can I justify this? $\wedge$ is stronger than $\to$. So shouldn't it be $p \to (r \wedge q) \to r$?

Read according to a plausible convention about repeated $\to$, this is $(p \to (r \wedge q)) \to r$. But few people would write it in this way.

So $p \to r \wedge q \to r$ seems more likely to mean $(p \to r) \wedge (q \to r)$.

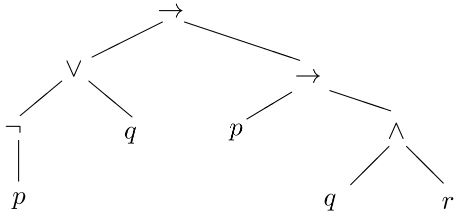
> Formulas like $p \to r \wedge q \to r$ can be misinterpreted without brackets, even if they are not strictly needed. The lesson: Don't write such formulas without brackets. USE BRACKETS if it helps readability, even if they are not strictly needed.

Parsing: formation tree, logical form

We have shown how to read a formula unambiguously — to parse it.

The information we gain from this can be represented as a tree: the *formation tree* of the formula.

For example, $\neg p \vee q \to (p \to q \wedge r)$ has the formation tree:



This is a nicer (but too expensive) way to write formulas.

Principal connective

Note that the connective at the root (top!) of the tree is $\to$. This is the *principal connective* or *main connective* of $\neg p \lor q \to (p \to q \land r)$.

This formula has the *overall* *logical form* $\phi \to \psi$.

Every non-atomic formula has a principal connective, which determines its *overall* logical form. You have to learn to recognise it.

- $p \land q \to r$ has principal connective $\to$.
  Its *overall* logical form is $\phi \to \psi$.
- $\neg(p \to \neg q)$ has principal connective $\neg$.
  Its *overall* logical form is $\neg\phi$.
- $p \land q \land r$ has principal connective $\land$ (probably the 2nd one!). Its logical form is $\phi \land \psi$.

> I stress *overall* as, in general, a formula has more than one logical form. For example, $\neg(p \to \neg q)$ has the logical forms: $\neg\phi$, $\neg(\phi \to \psi)$ — and according to most definitions, also just ϕ.

Subformulas

The tree view makes it easy to explain what a subformula is.

The *subformulas* of a formula ϕ are the formulas built in the stages on the way to building ϕ as in Definition 1.1.

They correspond to the nodes, or to the subtrees, of the formation tree of ϕ.

The subformulas of $\neg p \vee q \to (p \to q \wedge r)$ are:

$$\neg p \vee q \to (p \to q \wedge r)$$

$$\neg p \vee q \qquad\qquad\qquad p \to q \wedge r$$

$$\neg p \qquad\quad q \qquad\quad p \qquad\quad q \wedge r$$

$$p \qquad\qquad\qquad\qquad\qquad q \qquad\quad r$$

There are *two different* subformulas p. And $p \vee q$ and $p \to q$ are substrings but NOT subformulas.

Abbreviations

You may see some funny-looking formulas in books:

- $\bigwedge_{1 \le i \le n} \phi_i$, $\bigwedge_{i=1}^{n} \phi_i$,
  These all abbreviate $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$.
  Example: $\displaystyle\bigwedge_{1 \le i \le n} p_i \to q$ abbreviates $p_1 \wedge \ldots \wedge p_n \to q$.
- $\bigvee_{1 \le i \le n} \phi_i$, $\bigvee_{i=1}^{n} \phi_i$, $\displaystyle\bigvee_{i=1}^{n} \phi_i$, $\displaystyle\bigvee_{1 \le i \le n} \phi_i$, $\displaystyle\bigvee\!\!\!\!\bigvee_{1 \le i \le n} \phi_i$.
  These abbreviate $\phi_1 \vee \phi_2 \vee \ldots \vee \phi_n$.

  This is like in algebra:

$$\sum_{i=1}^{n} x_i \text{ abbreviates } x_1 + x_2 + \cdots + x_n.$$

Technical terms for logical forms

To understand logic books, you'll need some jargon. Learning it is tedious but necessary.

Definition 1.2

- A formula of the form $\top$, $\bot$ or p for an atom p, is (as we know) called *atomic*.
- A formula whose logical form is $\neg\phi$ is called a *negated formula*. A formula of the form $\neg p$, $\neg\top$, or $\neg\bot$ is sometimes called *negated-atomic*.
- A formula of the form $\phi \wedge \psi$ is called a *conjunction* and ϕ, ψ are its *conjuncts*.
- A formula of the form $\phi \vee \psi$ is called a *disjunction*, and ϕ, ψ are its *disjuncts*.
- A formula of the form $\phi \to \psi$ is called an *implication.* ϕ is called the *antecedent*, ψ is called the *consequent*.
- A formula of the form $\phi \leftrightarrow \psi$ is called a *bidirectional implication.*

Technical terms ctd. — literals and clauses

Definition 1.3 (Literals and clauses)

– A formula that is either atomic or negated-atomic is called a *literal*.

– A *clause* is a disjunction ($\vee$) of one or more literals.

E.g., the formulas p, $\neg r$, $\neg\bot$, $\top$ are all literals.

The following are all clauses.

$$p \qquad\qquad p \vee \neg q \vee r$$

$$\neg p \qquad\qquad p \vee p \vee \neg p \vee \neg\bot \vee \top \vee \neg q$$

How about $p \vee \neg q \wedge \neg p$?

Some people allow the empty clause (the disjunction of zero literals), which by convention is treated the same as $\bot$.

1.2 Semantics — The meaning

The Boolean connectives ($\wedge$ `and`, $\neg$ `not`, $\vee$ `or`, $\rightarrow$ `if-then`, $\leftrightarrow$ `if-and-only-if`) have *roughly* their English meanings.

But English is a natural language, full of ambiguity, subtlety, and special cases.

We need a precise idea of the meaning of formulas:

- especially as there are infinitely many of them,
- because we may want to implement it.

In propositional logic, our concern is to study good ('valid') inferences which depend just on:

- the truth or falsity of the propositional atoms; and
- the logical form in terms of Boolean connectives.

Atomic evaluation functions

A *situation* is simply something that determines truth-values, *true* (tt) or *false* (ff), to each propositional atom in the language. (Some write 1, 0 instead.) It is formally given by an *atomic evaluation function*.

Definition 1.4 (Atomic evaluation function)

Let $\mathcal{A}$ be a set of propositional atoms. An *atomic evaluation function* $v : \mathcal{A} \to \{\mathsf{tt}, \mathsf{ff}\}$ assigns truth-values to each propositional atom in $\mathcal{A}$.

There is more than one situation. In a different situation, the truth-values may be different.

Let v be an atomic evaluation function. The situation in which the atom p is true is one where we would use v with $v(p) = \mathsf{tt}$.

Evaluation functions

Knowing the situation, we can work out the truth-value of any given propositional formula — whether it is true or false in this situation.

Definition 1.5 (Evaluation function)

Let $\mathcal{A}$ be a set of propositional atoms, and v an atomic evaluation function for $\mathcal{A}$. The *evaluation function* $|\ldots|_v$ assigns the truth-value *true* (tt) or *false* (ff) to formulas as follows.

- If ϕ is an atom $p \in \mathcal{A}$: $|p|_v = \mathsf{tt}$ *iff* $v(p) = \mathsf{tt}$
- $|\neg\phi|_v = \mathsf{tt}$ *iff* $|\phi|_v = \mathsf{ff}$
- $|\phi \wedge \psi|_v = \mathsf{tt}$ *iff* $|\phi|_v = \mathsf{tt}$ and $|\psi|_v = \mathsf{tt}$
- $|\phi \vee \psi|_v = \mathsf{tt}$ *iff* $|\phi|_v = \mathsf{tt}$ or $|\psi|_v = \mathsf{tt}$
- $|\phi \rightarrow \psi|_v = \mathsf{tt}$ *iff* $|\phi|_v = \mathsf{ff}$ or $|\psi|_v = \mathsf{tt}$
- $|\phi \leftrightarrow \psi|_v = \mathsf{tt}$ *iff* $|\phi|_v = |\psi|_v$
- $|\bot|_v = \mathsf{ff}$ – $|\top|_v = \mathsf{tt}$

Note that the definition does not explicitly say when a formula evaluates to false. We don't add these since we know a formula is assigned ff *iff* it is not assigned the truth-value tt.

Evaluation functions — Examples

Suppose that $v(p) = \mathsf{tt}$ and $v(q) = \mathsf{ff}$. Then:

- $|\neg p|_v = \mathsf{ff}$
- $|\neg q|_v = \mathsf{tt}$
- $|p \wedge q|_v = \mathsf{ff}$
- $|p \vee q|_v = \mathsf{tt}$
- $|\neg p \wedge \neg q|_v = \mathsf{ff}$
- $|p \to \neg q|_v = \mathsf{tt}$

Do not confuse tt, ff (or 1, 0) with $\top$, $\bot$; the latter are formulas, not truth-values.

Now suppose that $v(p) = \mathsf{ff}$ and $v(q) = \mathsf{tt}$. Then:

- $|\neg p|_v = \mathsf{tt}$
- $|\neg q|_v = \mathsf{ff}$
- $|p \wedge q|_v = \mathsf{ff}$
- $|p \vee q|_v = \mathsf{tt}$
- $|\neg p \wedge \neg q|_v = \mathsf{ff}$
- $|p \to \neg q|_v = \mathsf{tt}$
- $|\top|_v = \mathsf{tt}$

We don't ask whether a formula is true. (It's like asking 'is 3x = 7 true?')
We ask if it is true in a given situation.

Understanding evaluation functions

Note that '$\vee$' means *inclusive* or: if both ϕ and ψ are true, then so is $\phi \vee \psi$. If you want 'exclusive' or, write: $(\phi \vee \psi) \wedge \neg(\phi \wedge \psi)$.

Consider "If there's smoke, then there's fire". If it's false that there's smoke, does that make the whole if-then claim true?

The semantics of '$\rightarrow$' we gave does make $\phi \rightarrow \psi$ true where ϕ is false. (Regardless of ψ, or of any causal relationship between ϕ and ψ.)

> Advice: Treat $\phi \rightarrow \psi$ as defined in Definition 1.5. It can often be used to model if-then claims; but all it asserts is the relationship between the truth-values of p and q as stated in the definition.

The evaluation function makes the Boolean connectives *truth functional*: the truth-value of the the whole formula is functionally determined by the truth-values of the "connected" subformulas.

Truth tables

Our connectives can also have their semantics given using *truth tables*. These give the same information as Definition 1.5, for *every possible* v.

Truth tables show how the truth-values of complex formulas depend on the truth-values of their subformulas.

p	$\neg p$
tt	ff
ff	tt

$\top$	$\bot$
tt	ff

ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \to \psi$	$\phi \leftrightarrow \psi$
tt	tt	tt	tt	tt	tt
tt	ff	ff	tt	ff	ff
ff	tt	ff	tt	tt	ff
ff	ff	ff	ff	tt	tt

Truth tables ctd.

Consider the situation where p is true (i.e., $v(p) = \mathsf{tt}$) and q is false (i.e., $v(q) = \mathsf{ff}$); this fixes what the atomic evaluation v must be.

In this situation, the formula $p \to q$ is assigned the truth-value ff (according to the truth table in in the previous slide.)

p	q	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
.	.	.	.	.	.
tt	ff	.	.	ff	.
.	.	.	.	.	.

Let's revisit Definition 1.5:

$$|p \to q|_v = \mathsf{tt} \quad \textit{iff} \quad |p|_v = \mathsf{ff} \ \textit{ or } \ |q|_v = \mathsf{tt}$$

$$\textit{iff} \quad \mathsf{tt} = \mathsf{ff} \ \textit{ or } \ \mathsf{ff} = \mathsf{tt} \quad \textit{(in our case)}$$

The RHS is false, so $|p \to q|_v = \mathsf{ff}$, as the truth table said.

Meanings of other connectives

Any truth table determines a connective. For instance:

ϕ	ψ	$\phi \uparrow \psi$
tt	tt	ff
tt	ff	tt
ff	tt	tt
ff	ff	tt

$\uparrow$, as defined here, is called the *Sheffer stroke*.

It has a special property: all our other Boolean connectives can be defined in terms of it.

Let's try to express $\neg p$ using only the Sheffer stroke operation $\uparrow$.

p	$\neg p$	$. \uparrow .$
tt	ff	.
ff	tt	.

Other connectives ctd.

It is often useful to know how many *truth-functionally*, distinct Boolean connectives of a given arity can be constructed.

There are *two* 0-ary connectives, and *four* 1-ary connectives. Why?

One way to see that there are four 1-ary connectives is to list the two truth assignments for an atom p, tt and ff.

Let Θ represent an arbitrary 1-ary connective. It can have any of the following truth tables.

p	Θp
tt	tt
ff	tt

p	Θp
tt	ff
ff	ff

p	Θp
tt	tt
ff	ff

p	Θp
tt	ff
ff	tt

In general, there are 2^{2^n} distinct n-ary Boolean connectives.

Functional completeness

Definition 1.6 (Functional completeness)

Let $\mathcal{C}$ be a set of Boolean connectives. $\mathcal{C}$ is functionally complete (for propositional logic) if any connective of any arity can be defined just in terms of the connectives in $\mathcal{C}$.

For instance:

The Sheffer stroke is functionally complete for propositional logic.

The set $\{\neg, \wedge\}$ is functionally complete for propositional logic (i.e., any connective, including $\uparrow$, can also be expressed with $\wedge$ and $\neg$).

$$\phi \uparrow \psi \text{ can be expressed as } \neg(\phi \wedge \psi)$$

1.3 English correspondence

Translating from English to logic is difficult. But we have to do it for applications.

We can only translate (some) *statements;* not questions, commands/invitations, exclamations.

For example,

I am king of the world	(‘I am king of the world’)
I bought milk and cookies	(‘I bought milk’ $\wedge$ ‘I bought cookies’)
There’s no crying in baseball	($\neg$ ‘There is crying in baseball’)
The answer is yes or no	(‘The answer is yes’ $\vee$ ‘the answer is no’)

Let us consider the following sentence.

The train is delayed and we don’t have a car

Translation in action

The train is delayed **and** we don't have a car

(The train is delayed **and** we don't have a car)

('The train is delayed' **and** we don't have a car)

('The train is delayed' **and it's not the case that** we have a car)

('The train is delayed' **and** **(it's not the case that** we have a car))

('The train is delayed' **and** **(it's not the case that** 'we have a car'))

Take d to represent 'The train is delayed' and c for 'we have a car', then the above English sentence is translated into the following propositional logic formula:

$$(d \wedge (\neg c))$$

English variants

– *'But'* means 'and'. (So does *yet*, *although*, *though*.)

E.g., I will go out, but it is raining
Let g be an atom for 'I will go out' and r for 'it is raining'. We get $(g \wedge r)$.

– *'Unless'* generally means 'or'.
E.g., I will go out unless it rains
Let g be an atom for 'I will go out' and w for 'it will rain'. (Note the extra 'will'.)
We get $(g \vee w)$. You could also use $((\neg w) \rightarrow g)$.

But you may think that 'I will go out unless it rains' implies that if it does rain then I won't go out. This is a *stronger* reading of 'unless'

– *Strong 'Unless'* (also called "exclusive or").

'I will go out unless it rains' becomes $(g \leftrightarrow (\neg w))$.

English variants ctd.

– *'Only if'*

You will pass only if your average is at least forty	('You will pass' $\rightarrow$ 'your average is at least forty')

– *'Is necessary for'*

James' attending the course is necessary for his obtaining a certificate of attendance	('James obtains a certificate of attendance' $\rightarrow$ 'James attends the course')

– *'Is sufficient for'*

Layla's timely arrival for rehearsals is sufficient for her taking part in the play	('Layla arrives on time for rehearsals' $\rightarrow$ 'Layla will take part in the play')

Other variants: *provided that*, *on the condition that*, *in case*, ...

Note: Use symbols for propositional atoms rather than strings in quotes. (I was lazy!)

From logic to English

Let p be 'It is raining' and q be 'I go out'',

$p \wedge q$	translates to	*It is raining and I go out*
$\neg p$	translates to	*It is not the case it is raining*
$p \to q$	translates to	*If it is raining then I go out*

You might think this is straightforward:

E.g., $p \to \neg q$ translates to   If it is raining then it is not the case I go out.

But there are problems. Complex formulas are hard to render naturally in English.

From logic to English ctd.

'$\to$' is a nightmare to translate.

The semantics of $\to$ works superbly for logic.

But in English we use 'if-then' in many different ways, and not always carefully.

Don't read $\to$ as '*causes*' — too strong.

E.g., 'I play in league 2 football' $\to$ 'I play in the Premiere league' is true here and now.

But would you say I play in league 2 football , then I play in the Premiere league is true?

Modalities (pitfalls)

A modality can be seen as shifting the context of evaluation of an utterance.

Time:

'I will be rich and famous'. It can't be translated as
'I will be rich' and 'I will be famous.'

The formula is true if I get famous only after I lose all my money. The English probably isn't: it suggests I'll be rich and famous at the same time.

Permission:

'You can have chicken or fish'. It usually means
'You can have chicken' $\wedge$ 'you can have fish'.

Obligation:

'I must do topics or a foreign language'. It can't be translated as
'I must do topics $\vee$ 'I must do a language'
(was false — you could choose which to take).

See the 4th year module Modal Logic for Strategic Reasoning in AI.

A really vicious example

This one deserves careful thought. Consider the formulas:

1. $p \wedge q \to r$
2. $(p \to r) \vee (q \to r)$.

Let p be 'I throw the rock at the bottle', q be 'you throw the rock at the bottle', and r be 'the bottle shatters'.

> Then (1) says that **If** we both throw the rock at the bottle, **then** the bottle shatters.
>
> And (2) says If I throw the rock at the bottle **or** you throw it then the bottles shatters.

Right? Would you say these mean the same? I guess not.

But they do! The formulas (1) and (2) are true in exactly the same situations. They are '*logically equivalent*'! We'll see this later.

I think the trouble is that in the English version of (2), you read the 'or' as 'and'. You shouldn't!

Be mindful ...

1. 'Arron and Russell are students'.

   This could be rewritten as: 'Arron is a student' **and** 'Russell is a student'.

How about

2. 'Arron and Russell are friends'.

   Is this the same as: 'Arron is a friend' **and** 'Russell is a friend'?

We might argue our way to represent this by rephrasing the English sentence as

'Arron is a friend of Russell' **and** 'Russell is a friend of Arron'.

What about 'Arron is a friend of everyone'? FOL deals with these ...