Lecture 2:
The Syntax of Propositional Logic

Aims:

- To discuss what we mean by a logic;
- To look at what is meant by propositional logic;
- To look at the syntax of propositional logic.

2.1 Propositional Logic

- In this section of the course, we introduce logic in general, and then look at one particular logic, *propositional logic*. We use this to introduce many of the main ideas that are to be found in every logic. We will look at the syntax and semantics of this logic, how to translate simple sentences of English into syntactically well-formed expressions of this logic, and how to show the validity of arguments in a number of ways.
- You are warned that logic terminology and notation is quite variable between different books. I have been consistent within these notes, and have also tried to point out cases where books use different terminology and notation.
- Logic is the study of reasoning.
- It is used to study arguments, comprising premisses and conclusions, such as the following:

  1. Every elephant is grey.
  2. Clyde is an elephant.
  3. Therefore, Clyde is grey.

  This a valid argument.

  Why? Because: If all the premisses are true, then the conclusion must be true. This definition of a valid argument says nothing about whether the premisses are in fact true: their truth is not *necessary* for the validity of the argument, as witnessed by the validity of the following argument (despite the falsity of its first premiss):

  1. Every elephant is pink.
  2. Clyde is an elephant.
  3. Therefore, Clyde is pink.

  Nor is truth of the premisses *sufficient* for validity of an argument, as witnessed by the invalidity of the following (despite the truth of both premisses):

1. Every elephant is grey.
2. Martie is grey.
3. Therefore, Martie is an elephant.

(Martie could be a grey mouse, for example.)

- The validity of an argument rests on its *form*, not the truth of the sentences that comprise it.
- Logic is widely used in many areas of Computer Science.
  - Circuit design
  - Boolean data types
  - Program specification
  - Program verification
  - Logic programming
  - Automated theorem-proving
  - Artificial Intelligence
- A well-defined, precise, unambiguous notation is needed, leading us to what is referred to as *symbolic logic*.

  In fact, there is a large variety of symbolic logics. The two most common symbolic logics are:
  - Propositional logic
  - First-order predicate logic
- Since symbolic logic tries to formalise everyday reasoning to some extent, there will be connections between the logics we study and English, which we will note as they arise. However, symbolic logics are formal languages and so the connection will be less than perfect. Therefore, you must learn the formal semantics of the language. Students who try to 'get by' using informal paraphrases will regularly be led astray.

2.2 Statements

- Propositional logic (like most varieties of logic) is only concerned with sentences that make *statements*. (This is not a serious restriction: these are the sentences most often used in arguments and in the sort of descriptions we would use to specify a software system, for example.) Statements state a fact, or describe a state of affairs. Such sentences in English are also called 'declarative' or 'indicative' sentences. E.g.:
  - It is sunny.
  - Roses are red and violets are blue.
  - Nobody knows the answer to that question.
- So, we exclude questions, commands, requests, instructions, and so on. E.g.:
  - Is it sunny?
  - $x := x + 1$;
  - Get off my case!
  - I promise to be there.

  - I name this ship *“Dignity”*.

- Here are more examples of statements:
  - $44/2 = 11 \times 2$
  - $4 < 5$
  - $4 > 5$
  - For all integers x, $x + 0 = x$

- A statement will be **true** or **false**. It must be one or the other. (It can’t be both, nor can it be undefined or some halfway house). You can see we are already diverging from English, in which statements of less than determinate truth value are commonplace. **true** and **false** are referred to as *truth values.*

- We begin with *atomic statements.*

  E.g.

  Clyde is cold.

- In propositional logic, we don’t break the atomic statements down any further: we do not determine the truth values of the atomic statements by decomposing them into subexpressions that denote objects, properties and relationships. We simply stipulate whether the atomic statements are true or false. An atomic statement which is subjected to this level of analysis is called a *proposition.* (When logicians go on to break the atomic statements down, then they need a more expressive logic, called *(first order) predicate logic.*)

- We form *compound statements* from atomic statements, by joining them using *connectives.*

  E.g.

  Clyde is cold and Clyde is wet.

- This is a compound statement: the word ‘and’ is an example of a connective. The atomic statements are ‘Clyde is cold’ and ‘Clyde is wet’.

- We make a further simplification, one which makes us diverge further from English.

  The meaning (truth value) of a compound statement depends *only* on the meanings (truth values) of the statements of which it is composed.

- We must therefore restrict our attention to connectives that are *truth-functional.*

  ‘Clyde is cold and Clyde is wet.’

  Knowing the truth value of ‘Clyde is cold’ and of ‘Clyde is wet’ allows us to determine the truth value of ‘Clyde is cold and Clyde is wet’.

  ‘Clyde is cold because Clyde is wet.’

  Knowing the truth value of ‘Clyde is cold’ and of ‘Clyde is wet’ would still not be enough for us to determine the truth value of ‘Clyde is cold because Clyde is wet’: we would still know nothing about the reasons for his coldness.

- So, ‘and’ is a truth-functional connective but ‘because’ is not. (At least, as far as these examples show.)

  This restriction gives simplicity and yet logics that use only truth-functional connectives have proved very productive.

- An atomic proposition may just as well be represented by a name such as p, p_1 or q.

  These names are called *propositional symbols*.

  If desired, they can be related to English statements, as follows:

$$p =_{\text{def}} \text{Clyde is cold}$$
$$q =_{\text{def}} \text{Clyde is wet}$$

- We now look at the syntax and semantics of propositional logic.

2.3 The Syntax of Propositional Logic

- The expressions of this language are called **well-formed formulas** or **wffs**.
- I'll define the syntax in two ways, first using an informal recursive definition and then using a BNF-grammar. Obviously, one definition is enough. The only reason for showing you two definitions is in case you are more comfortable with one over the other.
- So, here's the recursive definition.

  1. Every *propositional symbol* is a wff. (We use p, p_1, p_2, p_3,..., q, q_1, q_2, q_3... as propositional symbols.)
  2. For any wff W, the following is also a wff:

     $$\neg W$$

  3. For any two wffs W_1 and W_2, the following are also wffs:

     $$W_1 \wedge W_2 \qquad W_1 \vee W_2$$
     $$W_1 \Rightarrow W_2 \qquad W_1 \Leftrightarrow W_2$$

  4. Any wff may be enclosed in parentheses and will still be a wff.
  5. No other string of symbols is a wff.

- Rules (2) and (3) form compound wffs from other wffs.
- The symbols $\neg, \wedge, \vee, \Rightarrow$ and $\Leftrightarrow$ are the *connectives*.
- Here are examples of wffs:

$$\begin{array}{ccc} p & q & p_1 \\ (\neg p) & (\neg q) & (\neg(\neg q)) \\ (p \wedge p) & (p \vee q) & (p_1 \Rightarrow (\neg q_1)) \\ (p \Leftrightarrow (q_1 \vee q_2)) & ((p_1 \wedge p_2) \vee (q_1 \wedge q_2)) & (\neg((p_1 \wedge p_2) \vee (q_1 \wedge q_2))) \end{array}$$

- Here's the same language, defined using BNF.

$$
\begin{aligned}
\langle\text{wff}\rangle &::= (\ \langle\text{wff}\rangle\ ) \\
\langle\text{wff}\rangle &::= \neg\ \langle\text{wff}\rangle \\
\langle\text{wff}\rangle &::= \langle\text{wff}\rangle \land \langle\text{wff}\rangle \\
\langle\text{wff}\rangle &::= \langle\text{wff}\rangle \lor \langle\text{wff}\rangle \\
\langle\text{wff}\rangle &::= \langle\text{wff}\rangle \Rightarrow \langle\text{wff}\rangle \\
\langle\text{wff}\rangle &::= \langle\text{wff}\rangle \Leftrightarrow \langle\text{wff}\rangle \\
\langle\text{wff}\rangle &::= \langle\text{prop symbol}\rangle \\
\langle\text{prop symbol}\rangle &::= p \\
\langle\text{prop symbol}\rangle &::= p_0 \\
\langle\text{prop symbol}\rangle &::= p_1 \\
\langle\text{prop symbol}\rangle &::= p_2 \\
\langle\text{prop symbol}\rangle &::= q \\
\langle\text{prop symbol}\rangle &::= q_0 \\
\langle\text{prop symbol}\rangle &::= q_1 \\
\langle\text{prop symbol}\rangle &::= q_2 \\
\vdots\ &\ \vdots\ \ \vdots
\end{aligned}
$$

- Both definitions (recursive and BNF) are ambiguous.

  When it comes to the semantics of a wff, we will need to know which subwffs to evaluate first.

  - Parentheses determine evaluation order.
  - Precedence rules can be given:

    $$\text{Highest } \neg \lor \land \Rightarrow \Leftrightarrow \text{ Lowest}$$

    e.g. $p \lor q \land r \Rightarrow p$ evaluates as $((p \lor q) \land r) \Rightarrow p$
  - Associativity rules can be given:

    $$\begin{aligned} \text{Associate to the left:} &\quad \land \lor \Leftrightarrow \\ \text{Associate to the right:} &\quad \Rightarrow \end{aligned}$$

    e.g. $p \land q \land r \Rightarrow p \Rightarrow r$ evaluates as $((p \land q) \land r) \Rightarrow (p \Rightarrow r)$

- On the whole, in this module we will use a lot of parentheses. But, you must remember that $\neg$ has highest precedence. This is very useful in reducing the number of parentheses. So, e.g., $\neg p \land q$ evaluates as $(\neg p) \land q$, and, for the other evaluation order, you would write $\neg(p \land q)$.