Logic, propositions and representation of information

Logic is the basis of all mathematical reasoning. The rules of logic specify the meaning of mathematical statements. Logic has numerous applications in Computer Science, varying from design of digital circuits, to the construction of computer programs and verification of correctness of programs.

A proposition is the basic building block of logic:

Definition 1. A **proposition** is a statement that is either TRUE (denoted **T**) or FALSE denoted (denoted **F**).

Example 2. The following are propositions:

- $2 + 3 = 5$
- $1 + 2 = 4$
- 7 is an even number
- All dogs are brown

The following are not propositions:

- How far is it to the next town?
- Clean up your room!

In English, we can modify, combine, and relate propositions with words such as "not", "and", "or" and "if-then".

Example 3. (a) Consider the meaning of "and" in the following sentences:

- He entered the room and sat down ('then')
- She bought a computer and a printer ('and')
- Students in classes 101 and 202 ('or')

(b) Question: "Would you like a beer or a whisky" means an exclusive or: "either or" – both would be impolite:

(c) Double negative: "We doN'T need NO education"

The logical use of "and", "or", "not", "if-then" can be quite different from how these are used in natural language. The use in natural language is more free, but in logic they are defined very precisely. We use **truth tables** to introduce the **logical connectives**. A truth table indicates the TRUE/FALSE value of the combined proposition for each combination of the "participating proposition".

NOT: If P is a proposition, then the proposition NOTP is often denoted by $\neg P$. It is defined by the following truth table:

P	$\neg P$
T	**F**
F	**T**

Note that $\neg\neg P$ is the same as P. Note also that $\neg(x \in X)$ is the same as $x \notin P$.

AND: For two arbitrary propositions P and Q, the truth table for the proposition P AND Q is given by the following table. Note that P AND Q is often written $P \wedge Q$.

P	Q	$P \wedge Q$
F	**F**	**F**
T	**F**	**F**
F	**T**	**F**
T	**T**	**T**

OR: For propositions P and Q, the proposition P OR Q is written $P \vee Q$. Its truth table is the following:

P	Q	$P \vee Q$
F	**F**	**F**
T	**F**	**T**
F	**T**	**T**
T	**T**	**T**

XOR: For propositions P and Q, the proposition P XOR Q. Its truth table is the following:

P	Q	P XOR Q
F	**F**	**F**
T	**F**	**T**
F	**T**	**T**
T	**T**	**F**

The word XOR is a shorthand for *exclusive or* and, in fact, XOR is close to what "or" means in natural languages.

IMPLIES: The connecting word "implies" is typically denoted by an arrow $\Rightarrow$.

P	Q	$P \Rightarrow Q$
F	**F**	**T**
T	**F**	**F**
F	**T**	**T**
T	**T**	**T**

The meaning of $P \Rightarrow Q$ is to describe **if-then** sentences/conditions representing that the truth of a proposition Q is dependent on the truth of the proposition P. For instance:

If you practice for an exam, **then** you will succeed

can be interpreted to be an implication $P \Rightarrow Q$ such that $P =$ "if you practice for an exam" and $Q =$ "then you will succeed".

This implication is true:

If $x = 2$, **then** $2x = 4$.

The following example demonstrates even stranger side of implications:

If pigs can fly, **then** $2 + 2 = 5$

Let us denote $P =$ "pigs can fly" and $Q =$ "$2 + 2 = 5$". Pigs cannot fly, so the proposition P is false. Therefore, the implication $P \Rightarrow Q$ is true!

Here's an example of a false implication:

If the moon shines white, **then** the moon is made of white cheddar

Yes, the moon shines white. But the moon is *not made of white cheddar cheese*. This means that $P = \mathbf{T}$ and $Q = \mathbf{F}$ in implication $P \Rightarrow Q$. Therefore, the proposition is false.

An implication is true exactly when the if-part is false or the then-part is true

Let us consider the truth table of $\neg P \vee Q$:

P	Q	$\neg P$	$\neg P \vee Q$
F	**F**	**T**	**T**
T	**F**	**F**	**F**
F	**T**	**T**	**T**
T	**T**	**F**	**T**

The last column equals $P \Rightarrow Q$.

Example 4. Consider the following promise that a dad could give to a student:

"If you get 5/5 from the Mathematics A exam, then I'll give you a dollar'.'

The statement will be true if dad keeps his promise and false if he does not.

Suppose it is true that you get an 5/5 and it's true that dad gives you a dollar. Since dad kept his promise, the implication is true. This corresponds to the last line in the table.

Suppose it is true that you get 5/5 but it's false that dad gives you a dollar. Since he **did not** keep his promise, the **implication is false**. This corresponds to the second line in the table.

What if it is false that you get 5/5? Whether or not dad gives you a dollar, he has not broken his promise. Thus, the implication can't be false, so (since this is a two-valued logic) it must be true. This explains the first and the third lines of the table.

Remark 5. The *proposition* $P \Rightarrow Q$ is **defined** to be FALSE if P is TRUE and Q is FALSE. Otherwise, $P \Rightarrow Q$ is defined to be true TRUE. That's it! We need to remember that it is a *mathematical definition*, not everyday reasoning. Similarly, the interpreation of $\vee$ differs from the normal everyday usage of the word "or". The above examples of "flying pigs" and "dad giving you money" are just something which try to explain the implication. The connective $\Rightarrow$ is just a Boolean operation.

$P \Rightarrow Q$ does not mean *causality*: because pigs don't fly does not mean you **must go** to movie next week. Or because the moon is not made of cheese, this forces $2 + 2 = 4$. The proposition $P \Rightarrow Q$ is nothing more than another proposition built from propositions P and Q. This is a course on mathematics, not on philosophy.

You can try to seach, for instance, "false implies true" or "false implies anything" in the internet.

EQUIVALENCE: The proposition "P if and only if Q" asserts that P and Q are **logically equivalent**, that is, either both are true or both are false. The logical equivalence is denoted by $P \Leftrightarrow Q$.

P	Q	$P \Leftrightarrow P$
F	**F**	**T**
T	**F**	**F**
F	**T**	**F**
T	**T**	**T**

For example, the following if-and-only-if statement is true for all $x \in \mathbb{R}$:

$$x^2 - 4 \geq 0 \Leftrightarrow |x| \geq 2$$

For some values of x, both inequalities are true. For other values of x, both inequalities are false. Note also that many authors use the shorthand "iff" for "if and only if". This is not good English, but it is widely used.

A proposition is **tautology** if it is always true (the "last row" of the truth table consists of **T**-letters only). For example,

$$P \vee \neg P$$

is a tautology. On the other hand, the proposition is a **contradiction** if it is always false. For example

$$P \wedge \neg P$$

is a contradiction, because no proposition cannot be both true and false.

Note that if P and Q are logically equivalent, then $P \Leftrightarrow Q$ is a tautology. Similarly, if $P \Leftrightarrow Q$ is a tautology, then P and Q are logically equivalent.

Example 6. Consider these two sentences:

"If Lisa is in Finland, then Lisa is in Europe"

"If Lisa is not in Europe, then Lisa is not in Finland"

Let P be the proposition "Lisa is in Finland", and let Q be "Lisa is in Europe". The first sentence says "$P \Rightarrow Q$" and the second says "$\neg Q \Rightarrow \neg P$".

We can compare the statements $P \Rightarrow Q$ and $\neg Q \Rightarrow \neg P$ in a truth table:

P	Q	$P \Rightarrow Q$	$\neg Q$	$\neg P$	$\neg Q \Rightarrow \neg P$
F	**F**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**F**
F	**T**	**T**	**F**	**T**	**T**
T	**T**	**T**	**F**	**F**	**T**

This shows that $P \Rightarrow Q$ and $\neg Q \Rightarrow \neg P$ are logically equivalent.

The proposition $\neg Q \Rightarrow \neg P$ is called the **contrapositive** of the implication $P \Rightarrow Q$. The two are just different ways of saying the same thing. We will see later that in mathematical proofs, $P \Rightarrow Q$ is sometimes difficult to prove, but we are able to prove $\neg Q \Rightarrow \neg P$ instead.

Example 7. Consider that the following implications are true:

- If Sue is a programmer, then she is smart.
- If Sue is an early riser, then she does not like porridge.
- If Sue is smart, then she is an early riser.

We know that Sue likes porridge. Is she a programmer?

Example 8. The proposition $P \Leftrightarrow Q$ is logically equivalent to $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$. This can be seen in the truth table:

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$	$P \Leftrightarrow Q$
F	**F**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**F**
F	**T**	**T**	**F**	**F**	**F**
T	**T**	**T**	**T**	**T**	**T**

Example 9. We have the following correspondences between set theory and logical connectives: Let A and B be subsets of some universe U.

- $A \cup B = \{x \in U \mid x \in A \vee x \in B\}$
- $A \cap B = \{x \in U \mid x \in A \wedge x \in B\}$
- $A \ominus B = \{x \in U \mid x \in A \text{ XOR } x \in B\}$.
- $A^c = \{x \in U \mid \neg(x \in A)\}$
- $A \subseteq B$ if and only if $x \in A \Rightarrow x \in B$
- $A = B$ if and only if $x \in A \Leftrightarrow x \in B$

The programming language `Python` has the following operators which work for many data types:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These operations get truth values `True` or `False`.

Python also has the "connectives" `and`, `or`, `not`.

```
>>> 1 < 2 and 4 > 2
True

>>> 1 > 2 and 4 < 10
```

```
False

>>> not(1 > 2 and 4 < 10)
True

>>> 1 > 2 or 4 < 10
True
```

Using logical operations we can define **if-else selection**, whose general structure is the following:

```
if <condition>:
    <statement(s)>
else:
    <other statement(s)>
```

The `else`-part is not necessary. Here is an example of an if-else condition:

```
a = 2
b = 4

if a < b:
    print(a, 'is less than', b)
else:
    print(a, 'is not less than', b)
```

The program outputs: `2 is less than 4`. If we set `a = 5`, the program outputs: `5 is not less than 4`.

Another control structure is `while`-loop. Its idea is that we repeat something while a logical condition remains true. The

```
while <condition>:
    <statement(s)>
```

Here is a simple program using `while`-loop. It outputs the numbers $0, 1, \ldots, 9$ each of its own line.

```
n = 0
while n < 10:
    print(n)
    n = n + 1
```

Let us briefly consider how data is represented in a computer. A **bit** is the most basic unit of information in computing and digital communications. The name comes from "binary digit". One bit represents a logical state with two possible values. These values are most commonly represented as either "1" or "0", but they can be viewed also as the logical values TRUE and FALSE. Bits can be implemented in several forms. For instance, value of 1 (or TRUE) is represented by a more positive voltage relative to the representation of 0.

A **binary number** is a number expressed in binary system, which uses only the bits 0 and 1.

The usual *decimal counting* uses the ten symbols 0 through 9. Counting begins with the incremental substitution of the least significant digit (rightmost digit) which is often called the first digit. When the available symbols for this position are exhausted, the least significant digit is reset to 0, and the next digit of higher significance (one position to the left) is incremented (overflow), and incremental substitution of the low-order digit resumes. This method of "reset" and "overflow" is repeated for each digit of significance. Counting progresses as follows:

000, 001, 002, ..., 007, 008, 009,
010, 011, 012, ..., 017, 018, 019,
...
090, 091, 092, ..., 097, 098, 099,
100, 101, 102, ..., 107, 108, 109,
110, 111, 112, ..., 117, 118, 119,
...

Binary counting follows the same system, except that only the two symbols 0 and 1 are available. Thus, after a digit reaches 1 in binary, an increment resets it to 0 but also causes an increment of the next digit to the left:

0000, 0001,
0010, 0011,
0100, 0101,
0110, 0111,
1000, 1001,
1010, 1011,
1100, 1101,
1110, 1111

A decimal number D can be converted into binary numbers by the following method:

1. Divide the number D by 2.
2. Get the integer quotient for the next iteration as D

3. Get the remainder for the binary digit – remainder can be 0 or 1

4. Repeat the steps until the quotient is equal to 0.

The decimal number consists of the remainders such that the first remainder is the "rightmost"
Convert the decimal 13 to binary:

Div. by 2	Quotient	Remainder
13/2	6	1
6/2	3	0
3/2	1	1
1/2	0	1

This means that the binary representation of 13 is 1101.
Binary number with n digits:

$$d_{n-1} \cdots d_3 d_2 d_1 d_0$$

corresponds to the decimal number:

$$D = d_{n-1} \times 2^{n-1} + \cdots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times \underbrace{2^0}_{=1}$$

Note that $2^0 = 1$. Looking this representation, we also see a justification how the conversion from decimal to binary works.

If you divide D by 2, you get the quotient

$$D' = d_{n-1} \times 2^{n-2} + \cdots + d_2 \times 2^1 + d_1 \times 2^0$$

and the remainder is d_0, the rightmost desimal. If you divide D' by 2, the quotient is

$$D'' = d_{n-1} \times 2^{n-2} + \cdots + d_2 \times 2^0$$

and the remainder equals d_1, the second decimal from the right. This is repeated until nothing is left.

The binary number $1101 = d_3 d_2 d_1 d_0$ corresponds to the decimal

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13.$$

Binary addition is much like our normal everyday decimal addition, except that it carries on a value of 2 instead of a value of 10. In decimal addition, if you add 8 + 2 you get ten, which we write as 10. In the sum this gives a *sum* 0 and a *carrier* of 1. Something similar happens in binary addition when you add 1 and 1: the result is two (as always), but since two is written as 10 in binary, we get a *sum* 0 and a *carrier* of 1.
Therefore in binary:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$ (sum 0, carrier 1)

Let us us add 10 and 8 in binary. Decimal 10 corresponds the binary 1010 and 8 corresponds to 1000. We can sum binary numbers similarly as decimals:

```
  1 0 1 0
  1 0 0 0
----------
 1 0 0 1 0
```

The binary 10010 corresponds the decimal $2^1 + 2^4 = 2 + 16 = 18$.

Let us write this in the form of *truth table* so that 0 is interpreted as FALSE and 1 is interpreted as TRUE:

digit_1	digit_2	SUM	CARRIER
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

We can see that the column SUM corresponds to the logical connective XOR and the column CARRIER corresponds to AND.

An *integrated circuit* are carried on chips of silicon and consists of interconnected groups of transistors. An essential feature of transistors is that although subjected to continuously varying voltages, they may act as electrical switches (these details are not considered in this course). Transistors can be linked to to create **logic gates**, which recognize two levels of voltage: high (meaning 1) and low (meaning 0). We can then form *gates* which behave like AND, *or*, NOT, and XOR. This means that the problem of constructing a gate which specific characteristic is just that of finding a compound proposition with the desired output(s).

We end this section as considering sets as bit vectors. If the universe U has n elements a_1, a_2, ..., a_n then any set $A \subseteq U$ can be represented by a vector of n bits

$$(b_1, b_2, \ldots, b_n)$$

where the ith bit b_i is 1 if $a_i \in A$ and $b_i = 0$ if $a_i \notin A$.

Example 10. For instance, if $U = \{a, b, c, d\}$, then subsets of U can be represented as 4-bit vectors. For instance, if $A = \{a, d\}$ and $B = \{c, d\}$, then A corresponds to the vector $A = (1, 0, 0, 1)$ and $B = (0, 0, 1, 1)$. The empty set is $\emptyset = (0, 0, 0, 0)$ and the universe $U = (1, 1, 1, 1)$.

The set-operations *union*, *intersection*, *complement*, *difference* and *symmetric difference* can be performed by logical bit-wise operations. For instance, the union of sets A and B is the bit-wise OR (whose symbol is $\vee$). The intersection of sets A and B is the bit-wise AND, (whose symbol is $\wedge$):

$$A \cup B = (1 \vee 0, 0 \vee 0, 0 \vee 1, 1 \vee 1) = (1, 0, 1, 1)$$

and

$$A \cap B = (1 \wedge 0, 0 \wedge 0, 0 \wedge 1, 1 \wedge 1) = (0, 0, 0, 1).$$