8 Boolean Algebra

Logic is generally considered to lie in the intersection between Philosophy and Mathematics. It studies the meaning of statements and the relationship between them.

Logical statements in computer programs. Programming languages provide all the tools to be excessively precise. This includes *logical statements* which are used to construct loops, among other things. As an example, consider a while loop that exchanges adjacent array elements until some condition expressed by a logical statement is satisfied. Putting the while loop inside a for loop we get a piece of code that sorts an array $A[1..n]$:

```
for i = 1 to n do j = i;
  while j > 1 and A[j] > A[j − 1] do
    a = A[j]; A[j] = A[j − 1]; A[j − 1] = a;
    j = j − 1
  endwhile
endfor.
```

This particular method for sorting is often referred to as insertion sort because after $i-1$ iterations, $A[1..i-1]$ is sorted, and the i-th iteration inserts the i-th element such that $A[1..i]$ is sorted. We illustrate the algorithm in Figure 7. Here we focus on the logic that controls the while loop.

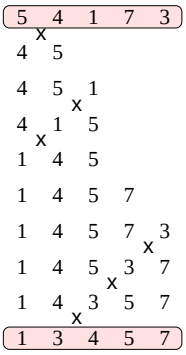

Figure 7: The insertion sort algorithm applied to an unsorted sequence of five integers.

The iteration is executed as long as two conditions hold, namely "$j > 1$" and "$A[j] > A[j-1]$". The first prevents we step beyond the left end of the array. The second condition limits the exchanges to cases in which adjacent elements are not yet in non-decreasing order. The two conditions are connected by a logical and, which requires both to be true.

Boolean operations. A logical statement is either true (T) of false (F). We call this the *truth value* of the statement. We will frequently represent the statement by a *variable* which can be either true or false. A *boolean operation* takes one or more truth values as input and produces a new output truth value. It thus functions very much like an arithmetic operation. For example, *negation* is a unary operation. It maps a truth value to the opposite; see Table 6. Much more common are binary operations; such as

p	$\neg p$
T	F
F	T

Table 6: Truth table for negation ($\neg$).

and, or, and exclusive or. We use a truth table to specify the values for all possible combinations of inputs; see Table 7. Binary operations have two input variables, each in one of two states. The number of different inputs is therefore only four. We have seen the use of these particular

p	q	$p \wedge q$	$p \vee q$	$p \oplus q$
T	T	T	T	F
T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

Table 7: Truth table for and ($\wedge$), or ($\vee$), and exclusive or ($\oplus$) operations.

boolean operations before, namely in the definition of the common set operations; see Figure 8.

$$\begin{aligned} A^c &= \{x \mid x \notin A\}; \\ A \cap B &= \{x \mid x \in A \text{ and } x \in B\}; \\ A \cup B &= \{x \mid x \in A \text{ or } x \in B\}; \\ A \oplus B &= \{x \mid x \in A \text{ xor } x \in B\}; \\ A - B &= \{x \mid x \in A \text{ and } x \notin B\}. \end{aligned}$$

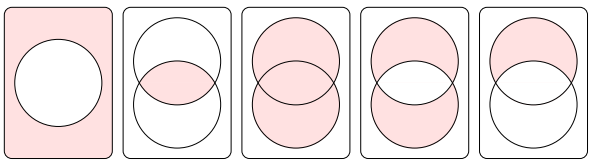

Figure 8: From left to right: the complement of one set and the intersection, union, symmetric difference, and difference of two sets.

Algebraic properties. We observe that boolean operations behave very much like ordinary arithmetic operations. For example, they follow the same kind of rules when we transform them.

- All three binary operations are commutative, that is,

$$\begin{aligned} p \wedge q \quad &\text{iff} \quad q \wedge p; \\ p \vee q \quad &\text{iff} \quad q \vee p; \\ p \oplus q \quad &\text{iff} \quad q \oplus p. \end{aligned}$$

- The and operation distributes over the or operation, and vice versa, that is,

$$\begin{aligned} p \wedge (q \vee r) \quad &\text{iff} \quad (p \wedge q) \vee (p \wedge r); \\ p \vee (q \wedge r) \quad &\text{iff} \quad (p \vee q) \wedge (p \vee r). \end{aligned}$$

Similarly, negation distributes over and and or, but it changes one into the other as it does so. This is known as de Morgan's Law.

DE MORGAN'S LAW. Letting p and q be two variables,

$$\begin{aligned} \neg(p \wedge q) \quad &\text{iff} \quad \neg p \vee \neg q; \\ \neg(p \vee q) \quad &\text{iff} \quad \neg p \wedge \neg q. \end{aligned}$$

PROOF. We construct the truth table, with a row for each combination of truth values for p and q; see Table 8. Since

p	q	$\neg(p \wedge q)$	$\neg p \vee \neg q$
T	T	F	F
T	F	T	T
F	T	T	T
F	F	T	T

Table 8: The truth table for the expressions on the left and the right of the first de Morgan Law.

the two relations are symmetric, we restrict our attention to the first. We see that the truth values of the two expressions are the same in each row, as required. □

Implications. The *implication* is another kind of binary boolean operation. It frequently occurs in statements of lemmas and theorems. An example is Fermat's Little Theorem. To emphasize the logical structure, we write A for the statement "n is prime" and B for "$a^{n-1} \bmod n = 1$ for every non-zero $a \in \mathbb{Z}_n$". There are different, equivalent ways to restate the theorem, namely "if A then B"; "A implies B"; "A only if B"; "B if A". The operation is

p	q	$p \Rightarrow q$	$\neg q \Rightarrow \neg p$	$\neg(p \wedge \neg q)$	$\neg p \vee q$
T	T	T	T	T	T
T	F	F	F	F	F
F	T	T	T	T	T
F	F	T	T	T	T

Table 9: The truth table for the implication ($\Rightarrow$).

defined in Table 9. We see the contrapositive in the second column on the right, which is equivalent, as expected. We also note that q is forced to be true if p is true and that q can be anything if p is false. This is expressed in the third column on the right, which relates to the last column by de Morgan's Law. The corresponding set operation is the complement of the difference, $(A - B)^c$; see Figure 9 on the left.

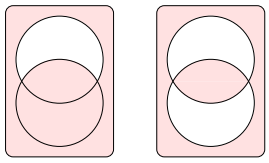

Figure 9: Left: the complement of the difference between the two sets. Right: the complement of the symmetric difference.

We recall that a logical statement is either true or false. This is referred to as the law of the *excluded middle*. In other words, a statement is true precisely when it is not false. There is no allowance for ambiguities or paradoxes. An example is the sometimes counter-intuitive definition that false implies true is true. Write A for the statement "it is raining", B for "I use my umbrella", and consider $A \Rightarrow B$. Hence, if it is raining then I use my umbrella. This does not preclude me from using the umbrella if it is not raining. In other words, the implication is not false if I use my umbrella without rain. Hence, it is true.

Equivalences. If implications go both ways, we have an *equivalence*. An example is the existence of a multiplicative inverse iff the multiplication permutes. We write A for the statement "a has a multiplicative inverse in $\mathbb{Z}_n$" and B for "the function $M : \mathbb{Z}_n \to \mathbb{Z}_n$ defined by $M(x) = a \cdot_n x$ is bijective". There are different, equivalent ways to restate the claim, namely "A if and only if B" and "A and B are equivalent". The operation is defined in Table 10. The last column shows that equivalence is the opposite of the exclusive or operation. Figure 9 shows the corresponding set operation on the right.

Recalling the definition of a group, we may ask which

p	q	$p \Leftrightarrow q$	$(p \Rightarrow q) \wedge (q \Rightarrow p)$	$\neg(p \oplus q)$
T	T	T	T	T
T	F	F	F	F
F	T	F	F	F
F	F	T	T	T

Table 10: The truth table for the equivalence ($\Leftrightarrow$).

of the binary operations form an Abelian group. The set is $\{\mathsf{F}, \mathsf{T}\}$. One of the two must be the neutral element. If we choose F then $\mathsf{F} \circ \mathsf{F} = \mathsf{F}$ and $\mathsf{F} \circ \mathsf{T} = \mathsf{T} \circ \mathsf{F} = \mathsf{T}$. Furthermore, $\mathsf{T} \circ \mathsf{T} = \mathsf{F}$ is necessary for T to have an inverse. We see that the answer is the exclusive or operation. Mapping F to 0 and T to 1, as it is commonly done in programming languages, we see that the exclusive or can be interpreted as adding modulo 2. Hence, $(\{\mathsf{F}, \mathsf{T}\}, \oplus)$ is isomorphc to $(\mathbb{Z}_2, +_2)$.

Summary. We have learned about the main components of logical statements, boolean variables and operations. We have seen that the operations are very similar to the more familiar arithmetic operations, mapping one or more boolean input variable to a boolean output variable.