1 Logical Operations

1.1 And

The "and" operator is a binary operator, denoted as $\wedge$, &, $\cdot$, or sometimes by just concatenating symbols, is true only if both parameters are true.

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

The expression "A and B" is often written as "$A \wedge B$" or "$A\&B$" or "$A \cdot B$" or "AB". The and operator is also known as a conjunction. "$A \wedge B$" and "AB" are the most common notations.

1.2 Or

The "or" operator is a binary operator, denoted as $\vee$, $|$, or $+$, is true if either parameter is true.

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

The expression "A or B" is often written as "$A \vee B$" or "$A|B$" or "$A + B$". The or operator is also known as a disjunction. All three notations are relatively common.

1.3 Not

The "not" operator is a unary operator, denoted as $\neg$, !, $-$, or as a line overtop the parameter, is true if the parameter is false.

A	$\neg A$
T	F
F	T

The expression "not A" is often written as "$\neg A$" or "$!A$" or "$-A$" or "$\overline{A}$". The not operator is also known as negation. All four notations are reasonably common, with "$!A$" mostly used in programming languages.

1.4 Exclusive-or

The exclusive-or operator is a binary operator, denoted as $\oplus$ or ˆ, is true if only one of its parameters is true. It is often abbreviated as xor, and pronounced "x-or".

A	B	$A \oplus B$
T	T	F
T	F	T
F	T	T
F	F	F

The expression “A xor B” is often written “$A \oplus B$” or “$A \hat{} B$”. The later notation is normally only seen in the context of C-derived programming languages such as C++ or Java.

1.5 Nand

The nand operator is a binary operator, denoted as “$\uparrow$” or “$\overline{\curlywedge}$” or as an and combined with a not, is false only when both parameters are true.

A	B	$A \uparrow B$
T	T	F
T	F	T
F	T	T
F	F	T

The expression “A nand B” is often written “$A \uparrow B$” or “$\neg(A \land B)$” or “$\overline{AB}$”. Nand is normally written as “$\neg(A \land B)$” or “$\overline{AB}$”.

1.6 Nor

The nor operator is a binary operator, denoted as “$\downarrow$” or “$\curlywedge$” or as an or combined with a not, is true only when both parameters are false.

A	B	$A \curlywedge B$
T	T	F
T	F	F
F	T	F
F	F	T

The expression “A nor B” is often written “$A \curlywedge B$” or “$\neg(A \lor B)$ or $\overline{A \lor B}$. Nor is normally written as “$\neg(A \lor B)$” or “$\overline{A \lor B}$”.

1.7 Implies

Logical implication is not always thought of as a binary operator, but it is. It is often written as “$A \implies B$” or “$A \to B$”. It is an important enough that the parameters to the operator have specific names as well. For “$A \implies B$”, A is the premise and B is the conclusion, alternatively known as the antecedent and consequent. An implication is true whenever the premise is false, or if both the premise and conclusion is true. When the premise is false, the implication referred to as being vacuously true.

A	B	$A \implies B$
T	T	T
T	F	F
F	T	T
F	F	T

Both forms “$A \implies B$” and “$A \to B$” are regularly used.

1.8 Biconditional

The biconditional operator, written as “$A \iff B$” or “$A \leftrightarrow B$”, is true only if the truth value of A and B are the same. It can be thought of as a conditional operator that goes both directions. It is often read as “if and only if” which is commonly abbreviated to “iff”

A	B	$A \iff B$
T	T	T
T	F	F
F	T	F
F	F	T

Both forms “$A \iff B$” and “$A \leftrightarrow B$” are regularly used.

2 Functional Completeness

Some of the operators just covered are redundant with respect to one another. In particular it should be obvious that one can express NAND with a combination of an AND and a NOT. More precisely, a set of operators is said to be `functionally complete` if it can express all possible truth tables. Note how with a binary operator there are four combinations of input truth values. Each one has a truth value, leading to $2^4 = 16$ possible truth tables.

There are two singleton operators that themselves form functionally complete sets, {NAND} and {NOR}. Other functionally complete sets include $\{AND, NOT\}$ and $\{AND, OR, NOT\}$.

3 Quantifiers

3.1 Universal

The universal quantifier is denoted as $\forall$ and read as “for all”. For example the expression

$$\forall A$$

is read as “for all A” and means that the statements is true if and only if it is true for every predicate A.

3.2 Existential

The existential quantifier is denoted as $\exists$ and read as “there exists”. For example the expression

$$\exists A$$

is read as “there exists A” and means that the statement is true if and only if there is some such predicate A that is true.

3.3 Unique Existential

The unique existential quantifier is denoted as $\exists!$ and read as “there exists a unique”. For example the expression

$$\exists! A$$

is read as “there exists a unique A” and means that the statement is true if and only if there is some such predicate A that is true, and there is no other predicate which is also true.

3.4 Negating Quantifiers

Negating quantifiers can generally be done as a mechanical process so long as the statement being negated is written formally.

$$\neg \forall P(x) \equiv \exists x \neg P(x)$$

$$\neg \exists P(x) \equiv \forall x \neg P(x)$$

3.5 Nesting Quantifiers

Quantifiers can be combined into one expression, and one needs to be careful about the interpretation.

For example, if presented with the expression

$$\forall x \forall y P(x, y),$$

it means that for the expression to be true, for each and every x, the statement $P(x, y)$ is true for all possible y and the statement is true for every x.

The expression

$$\forall x \exists y P(x, y)$$

means that for each x, there is a corresponding y, which may be different for each x, for which $P(x, y)$ is true and the statement is true for all x.

4 Set Theory

A set is a collection of elements. It could be finite in size, or infinite. Often sets are denoted using capital letters.

4.1 Set Membership

To denote that an element x belongs to a set A, it is written $x \in A$.

4.2 Subset

A set A is a subset of a set B if all elements in A are also in B. As a boolean expression, it is

$$A \subseteq B \iff \forall x \in A \implies x \in B.$$

A set A is a proper subset of a set B if A is a subset of B and the two sets are not the same.

$$A \subset B \iff \forall(x \in A \implies x \in B) \wedge A \neq B.$$

The notations shown above are common, but some authors use just $\subset$ to mean a regular subset and $\subsetneq$ to mean proper. It is also common to see subset as $\subseteq$ and $\subsetneq$ for proper subset. The exact combination used will vary, so make sure you know what exact combination is being used for whatever text you are working on.

4.3 Set Operators

Analogous to the various boolean operators, there are operators for sets that operate similarly.

4.3.1 Union

The union of two sets, denoted $A \bigcup B$, means the set that contains all elements from both A and B. Written as a boolean expression, it is

$$x \in A \bigcup B \iff x \in A \vee x \in B.$$

4.3.2 Intersection

The intersection of two sets, denoted $A \bigcap B$, means the set that contains all elements that are common to A and B. Written as a boolean expression, it is

$$x \in A \bigcap B \iff x \in A \wedge x \in B.$$

4.3.3 Difference

The difference of two sets, denoted $A - B$, means the set that contains all elements that are in A but not in B. Written as a boolean expression, it is

$$x \in A - B \iff x \in A \wedge x \notin B.$$

It is also written as $A \diagdown B$.

5 Common Sets

There are many common sets that we will be using during this course

- The integers. $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$.
- The natural numbers. $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

- The positive integers. $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$.
- The rational numbers. $\mathbb{Q} = \left\{\frac{a}{b} \colon a, b \in \mathbb{Z} \wedge b \neq 0\right\}$.
- The real numbers, $\mathbb{R}$. It is harder to write down a precise statement of the elements of the set of real numbers. For this course it is sufficient to think of them as any number that has a decimal expansion.

6 Usage of these operators in Programming

Several of the boolean operators discussed previously are used to perform bitwise operations when programming. The operators most often apply bit by bit to each bit within a data type.

- The AND operator is used to *mask* off bits. This works by observing that for $A \wedge B$, if B is zero, the result is always zero, which turns off the corresponding bit in the result. If B is one, it preserves whatever the bit is in A, regardless of what it is.
- The OR operator is used to *set* bits. This works by observing that for $A \vee B$, if B is zero the result is whatever A had for that bit. If B is one, the result is on, no matter what A is.
- The XOR operator is used to *toggle* bits. This works by observing that for $A \oplus B$, if B is zero, the result is whatever A had for that bit. If B is one, the result is toggling the corresponding bit in A.
- The NOT operator is used to *flip* all the bits. This is often used if it is easier to describe the complement to an operation instead of it directly. For example, due to some peculiarities in the C++ language, one often uses `0` instead of `0xF`... `FF` because the exact number of `Fs` needed can vary based on the platform used.