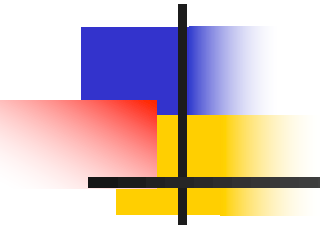
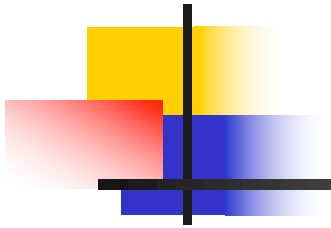


Mathematical notions and terminologies

Review of things we need later

Math toolbox



- Sets
- Sequences and Tuples
- Functions and Relations
- Graphs
- Strings and Languages
- Boolean Logic

More about functions

When the domain of a function f is $A_1 \times \cdots \times A_k$ for some sets $A_1, \ldots, A_k$, the input to f is a k-tuple $(a_1, a_2, \ldots, a_k)$ and we call the a_i the ***arguments*** to f. A function with k arguments is called a ***k-ary function***, and k is called the ***arity*** of the function. If k is 1, f has a single argument and f is called a ***unary function***. If k is 2, f is a ***binary function***. Certain familiar binary functions are written in a special ***infix notation***, with the symbol for the function placed between its two arguments, rather than in ***prefix notation***, with the symbol preceding. For example, the addition function add usually is written in infix notation with the $+$ symbol between its two arguments as in $a + b$ instead of in prefix notation $add(a, b)$.

Relations

A ***predicate*** or ***property*** is a function whose range is $\{\text{TRUE}, \text{FALSE}\}$. For example, let *even* be a property that is TRUE if its input is an even number and FALSE if its input is an odd number. Thus $even(4) = \text{TRUE}$ and $even(5) = \text{FALSE}$.

A property whose domain is a set of k-tuples $A \times \cdots \times A$ is called a ***relation***, a ***k-ary relation***, or a ***k-ary relation on A***. A common case is a 2-ary relation, called a ***binary relation***. When writing an expression involving a binary relation, we customarily use infix notation. For example, "less than" is a relation usually written with the infix operation symbol $<$. "Equality", written with the $=$ symbol, is another familiar relation. If R is a binary relation, the statement aRb means that $aRb = \text{TRUE}$. Similarly, if R is a k-ary relation, the statement $R(a_1, \ldots, a_k)$ means that $R(a_1, \ldots, a_k) = \text{TRUE}$.

Example

EXAMPLE 0.10

In a children's game called Scissors–Paper–Stone, the two players simultaneously select a member of the set {SCISSORS, PAPER, STONE} and indicate their selections with hand signals. If the two selections are the same, the game starts over. If the selections differ, one player wins, according to the relation *beats*.

beats	SCISSORS	PAPER	STONE
SCISSORS	FALSE	TRUE	FALSE
PAPER	FALSE	FALSE	TRUE
STONE	TRUE	FALSE	FALSE

From this table we determine that SCISSORS *beats* PAPER is TRUE and that PAPER *beats* SCISSORS is FALSE. ■

Equivalence relation

A special type of binary relation, called an ***equivalence relation***, captures the notion of two objects being equal in some feature. A binary relation R is an equivalence relation if R satisfies three conditions:

1. R is ***reflexive*** if for every x, xRx;
2. R is ***symmetric*** if for every x and y, xRy implies yRx; and
3. R is ***transitive*** if for every x, y, and z, xRy and yRz implies xRz.

Example

EXAMPLE 0.11

Define an equivalence relation on the natural numbers, written $\equiv_7$. For $i, j \in \mathcal{N}$, say that $i \equiv_7 j$, if $i - j$ is a multiple of 7. This is an equivalence relation because it satisfies the three conditions. First, it is reflexive, as $i - i = 0$, which is a multiple of 7. Second, it is symmetric, as $i - j$ is a multiple of 7 if $j - i$ is a multiple of 7. Third, it is transitive, as whenever $i - j$ is a multiple of 7 and $j - k$ is a multiple of 7, then $i - k = (i - j) + (j - k)$ is the sum of two multiples of 7 and hence a multiple of 7, too. ■

Math toolbox

Sets

Sequences and Tuples

Functions and Relations

Graphs

Strings and Languages

Boolean Logic

Graphs

An ***undirected graph***, or simply a ***graph***, is a set of points with lines connecting some of the points. The points are called ***nodes*** or ***vertices***, and the lines are called ***edges***, as shown in the following figure.

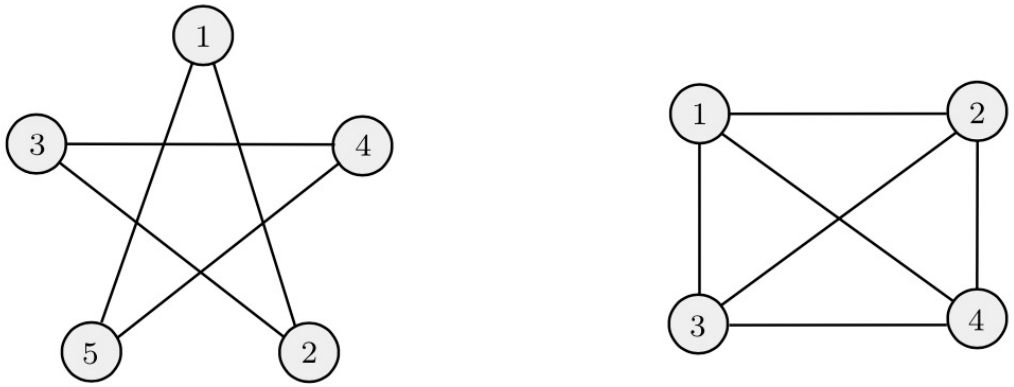


FIGURE 0.12
Examples of graphs

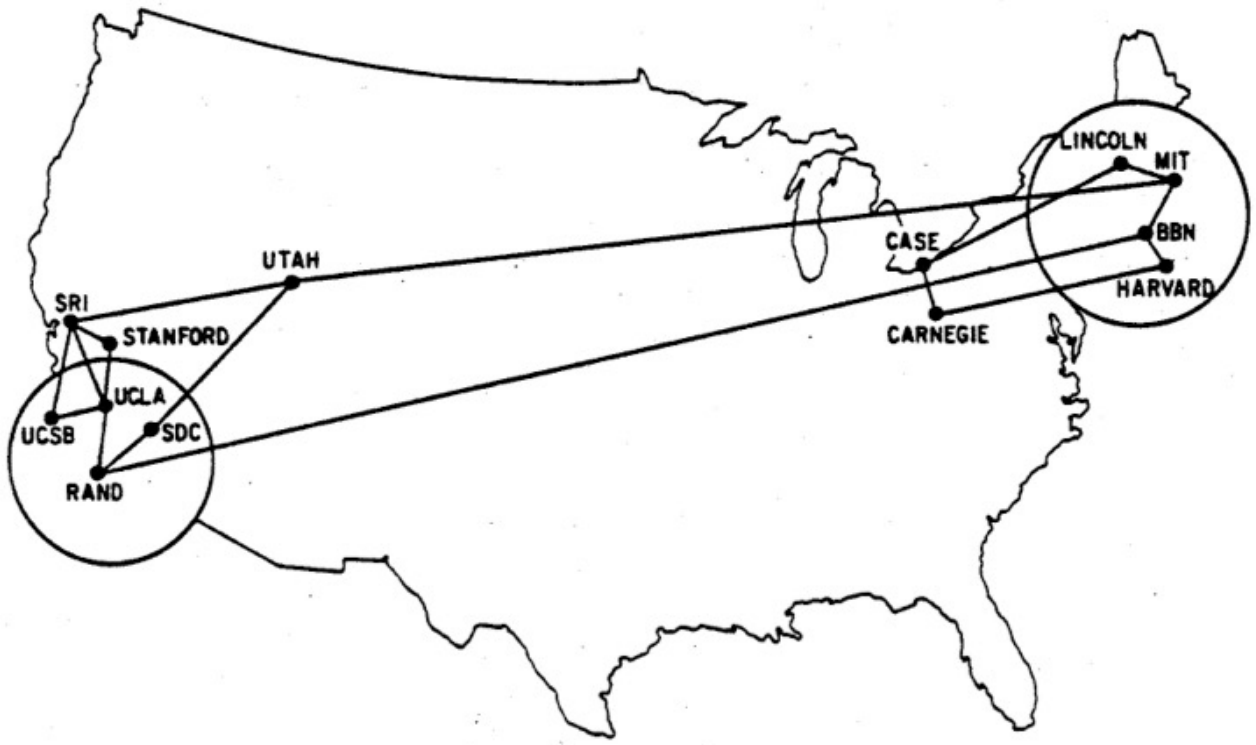
Graphs

The number of edges at a particular node is the ***degree*** of that node. In Figure 0.12(a), all the nodes have degree 2. In Figure 0.12(b), all the nodes have degree 3. No more than one edge is allowed between any two nodes. We may allow an edge from a node to itself, called a ***self-loop***, depending on the situation.

In a graph G that contains nodes i and j, the pair (i, j) represents the edge that connects i and j. The order of i and j doesn't matter in an undirected graph, so the pairs (i, j) and (j, i) represent the same edge. Sometimes we describe undirected edges with unordered pairs using set notation as in $\{i, j\}$. If V is the set of nodes of G and E is the set of edges, we say $G = (V, E)$. We can describe a graph with a diagram or more formally by specifying V and E. For example, a formal description of the graph in Figure 0.12(a) is

$$\big(\{1, 2, 3, 4, 5\},\ \{(1, 2),\ (2, 3),\ (3, 4),\ (4, 5),\ (5, 1)\}\big),$$

Side story: ARPANET (origin of internet)



Graphs as representations of data

Graphs frequently are used to represent data. Nodes might be cities and edges the connecting highways, or nodes might be people and edges the friendships between them. Sometimes, for convenience, we label the nodes and/or edges of a graph, which then is called a ***labeled graph***. Figure 0.13 depicts a graph whose nodes are cities and whose edges are labeled with the dollar cost of the cheapest nonstop airfare for travel between those cities if flying nonstop between them is possible.

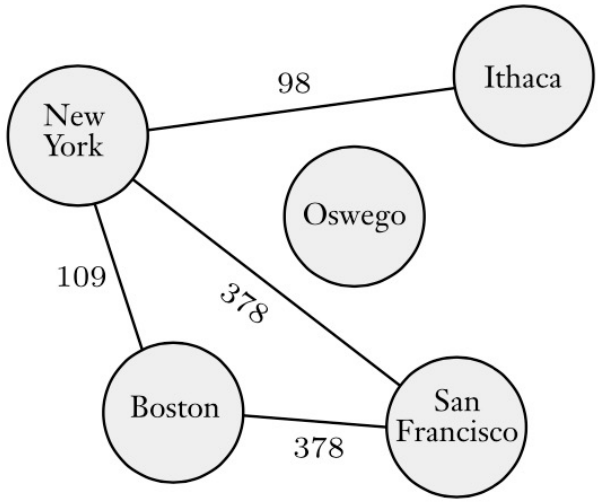
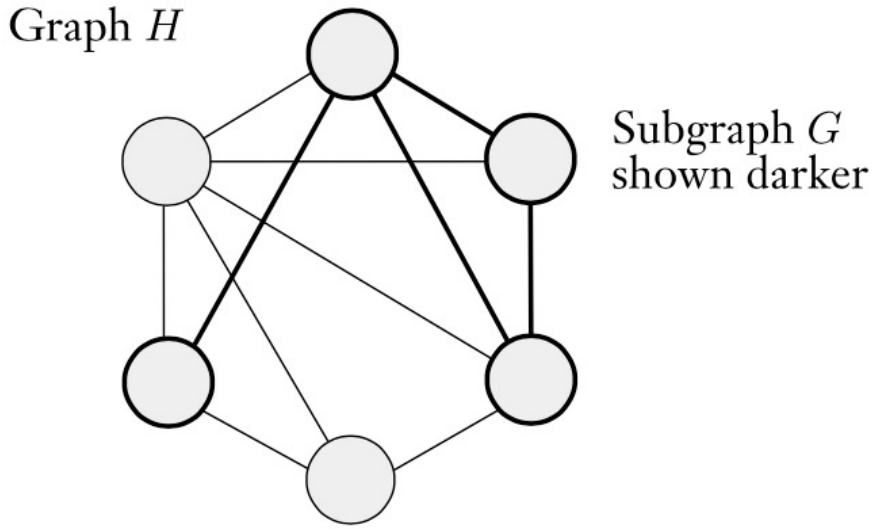


FIGURE 0.13
Cheapest nonstop airfares between various cities

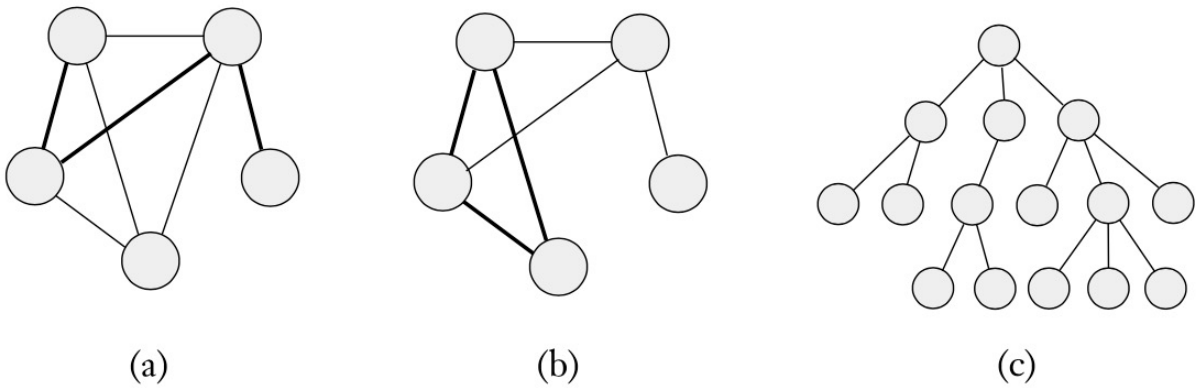
Subgraphs

We say that graph G is a ***subgraph*** of graph H if the nodes of G are a subset of the nodes of H, and the edges of G are the edges of H on the corresponding nodes. The following figure shows a graph H and a subgraph G.



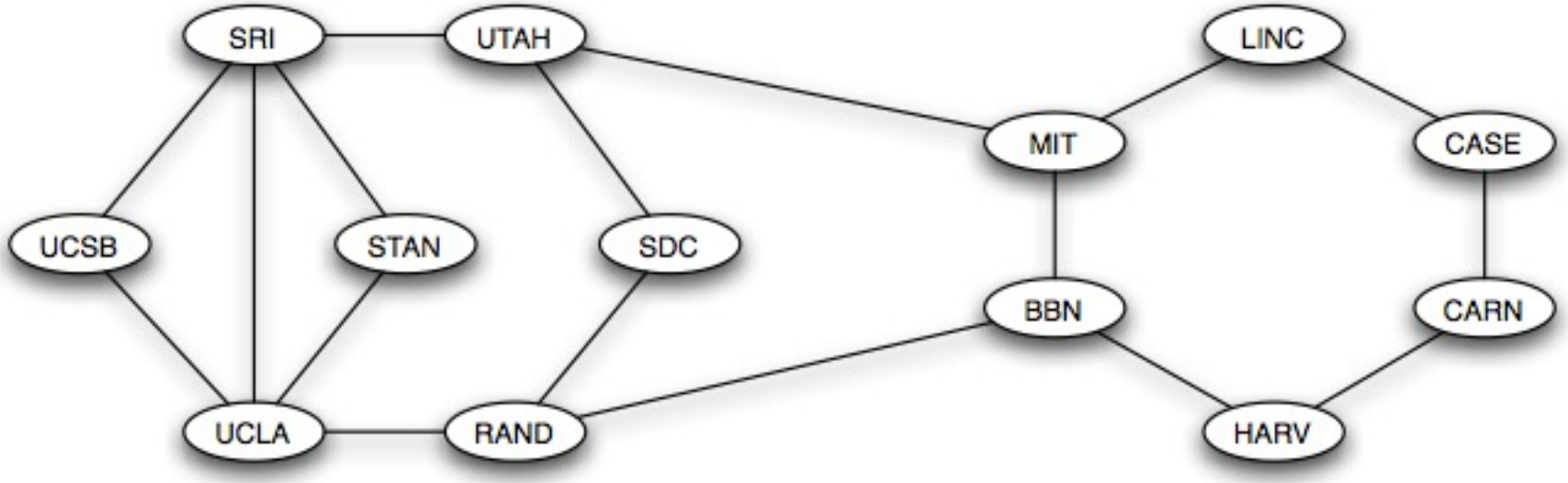
Paths and cycles

A ***path*** in a graph is a sequence of nodes connected by edges. A ***simple path*** is a path that doesn't repeat any nodes. A graph is ***connected*** if every two nodes have a path between them. A path is a ***cycle*** if it starts and ends in the same node. A ***simple cycle*** is one that contains at least three nodes and repeats only the first and last nodes. A graph is a ***tree*** if it is connected and has no simple cycles, as shown in Figure 0.15. A tree may contain a specially designated node called the ***root***. The nodes of degree 1 in a tree, other than the root, are called the ***leaves*** of the tree.



(a) (b) (c)

ARPANET revisited

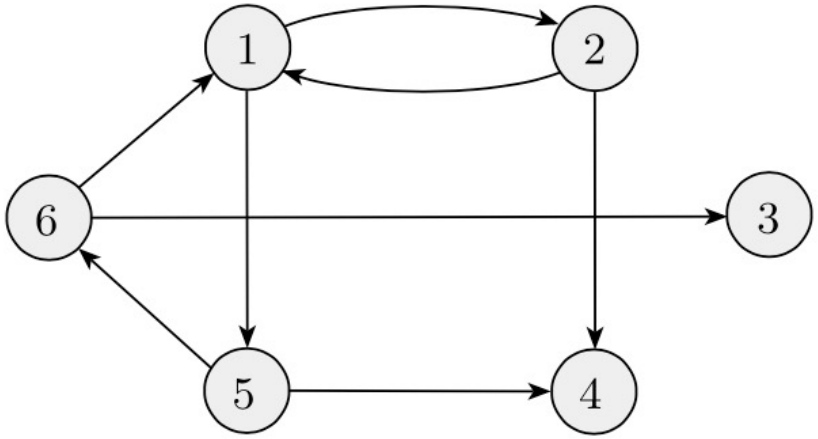


Every edge in the graph belongs to a cycle.

Why do you think it is designed to be so?

Directed graphs

A ***directed graph*** has arrows instead of lines, as shown in the following figure. The number of arrows pointing from a particular node is the ***outdegree*** of that node, and the number of arrows pointing to a particular node is the ***indegree***.

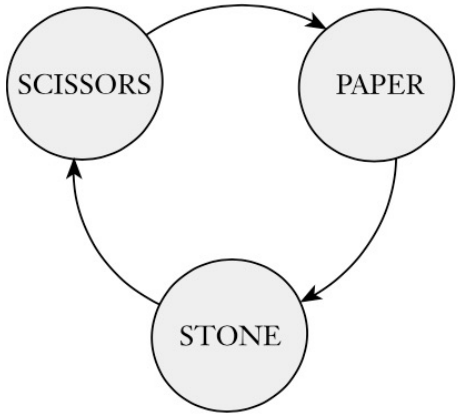


Directed graphs and binary relations

A path in which all the arrows point in the same direction as its steps is called a ***directed path***. A directed graph is ***strongly connected*** if a directed path connects every two nodes. Directed graphs are a handy way of depicting binary relations. If R is a binary relation whose domain is $D \times D$, a labeled graph $G = (D, E)$ represents R, where $E = \{(x, y) | \, xRy\}$.

EXAMPLE 0.17

The directed graph shown here represents the relation given in Example 0.10.



Math toolbox

Sets

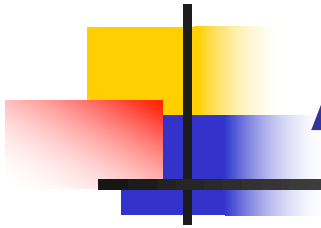
Sequences and Tuples

Functions and Relations

Graphs

Strings and Languages

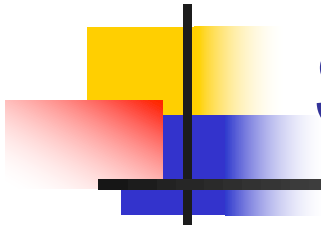
Boolean Logic



Alphabet

An alphabet is a finite, non-empty set of symbols

- We generally use capital Greek letters ∑ (sigma) and Γ (tau) to designate alphabets
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …



Strings

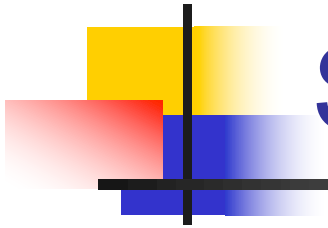
A string over an alphabet is a finite sequence of symbols from that alphabet

The symbols in a string are written next to one another and not separated by commas

Example:

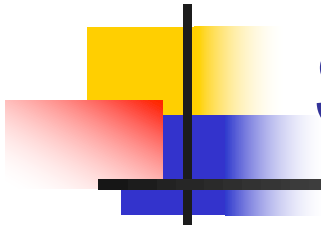
If $\Sigma_1 = \{0,1\}$ then 01001 is a string over Σ_1

If $\Sigma_2 = \{a, b, c, \ldots, z\}$, then tomato is a string over Σ_2



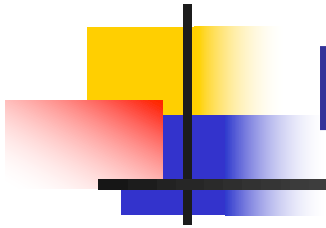
Strings

- The ***length*** of a string w, denoted by "$|w|$", is equal to the *number of symbols that it contains*
  - *E.g.,* $w = 010100$ $|w| = 6$
- The string of length zero is called ***the empty string*** and is written as ε
- The empty string plays the role of 0 in a number system
- If w has length n, we can write $w = w_1 w_2 \ldots w_n$ where each $w_i \in \Sigma$
- The reverse of of w, written w^R, is the string obtained by writing w in the opposite order
- String z is a substring of w if z appears consecutively within w.



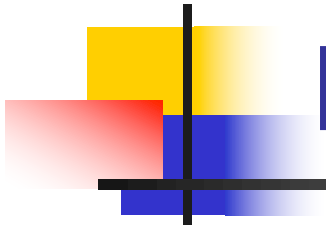
Strings

- **Concatenation**
  - If we have a string x of length m and string y of length n, the concatenation of x and y, written xy, is the string obtained by appending y to the end of x, as $x_1 \ldots x_m y_1 \ldots y_n$.
  - To concatenate a string with itself many times, we use the notation x^k to mean $xx \ldots x$ (k times)
- **Lexicographic order**
  - Same as familiar dictionary order
- **Prefix**
  - Say that string x is a prefix of string y if string z exists where $xz = y$
  - x is a proper prefix if in addition $x \neq y$
- **Language**
  - A language is a set of strings



Boolean logic

- Mathematical system built around the two values TRUE and FALSE
- Boolean operations
  - **Negation** (or NOT operation)
    - Example: ¬1 = 0
  - **Conjunction** (or AND operation)
    - The conjunction two Boolean values is 1 if both of those values are 1
    - Symbol: ∧
  - **Disjunction** (or OR operation)
    - The disjunction of two Boolean values is 1 if either of those values is 1
    - Symbol: ∨



Boolean logic

- **Exclusive or (XOR)**
  - Denoted by $\oplus$
  - Is 1 if either but not both of its operands is 1
- **Equality**
  - Denoted by $\leftrightarrow$
  - Is 1 if both of its operands have the same value
- **Implication**
  - Denoted by $\rightarrow$
  - Is 0 if its first operand is is 1 and its second operand is 0; otherwise $\rightarrow$ is 1

Boolean logic: summary

$0 \oplus 0 = 0$	$0 \leftrightarrow 0 = 1$	$0 \rightarrow 0 = 1$
$0 \oplus 1 = 1$	$0 \leftrightarrow 1 = 0$	$0 \rightarrow 1 = 1$
$1 \oplus 0 = 1$	$1 \leftrightarrow 0 = 0$	$1 \rightarrow 0 = 0$
$1 \oplus 1 = 0$	$1 \leftrightarrow 1 = 1$	$1 \rightarrow 1 = 1$

$P \vee Q$	$\neg(\neg P \wedge \neg Q)$
$P \rightarrow Q$	$\neg P \vee Q$
$P \leftrightarrow Q$	$(P \rightarrow Q) \wedge (Q \rightarrow P)$
$P \oplus Q$	$\neg(P \leftrightarrow Q)$

The ***distributive law*** for AND and OR comes in handy when we manipulate Boolean expressions. It is similar to the distributive law for addition and multiplication, which states that $a \times (b + c) = (a \times b) + (a \times c)$. The Boolean version comes in two forms:

- $P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$, and its dual
- $P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$.