

Propositional logic:

- *Propositional statement*: expression that has a truth value (true/false). It is a *tautology* if it is always true, *contradiction* if always false.
- *Logic connectives*: negation (“not”) $\neg p$, conjunction (“and”) $p \wedge q$, disjunction (“or”) $p \vee q$, implication $p \rightarrow q$ (equivalent to $\neg p \vee q$), biconditional $p \leftrightarrow q$ (equivalent to $(p \rightarrow q) \wedge (q \rightarrow p)$). The order of precedence: \neg strongest, \wedge next, \vee next, \rightarrow and \leftrightarrow the same, weakest.
- If $p \rightarrow q$ is an implication, then $\neg q \rightarrow \neg p$ is its *contrapositive*, $q \rightarrow p$ a *converse* and $\neg p \rightarrow \neg q$ an *inverse*. An implication is equivalent to its contrapositive, but not to converse/inverse or their negations. A negation of an implication $p \rightarrow q$ is $p \wedge \neg q$ (it is not an implication itself!)
- A *truth table* has a line for each possible values of propositional variables (2^k lines if there are k variables), and a column for each variable and subformula, up to the whole statement. The cells of the table contain T and F depending whether the (sub)formula is true for the corresponding values of variables.
- A *truth assignment* is a string of values of variables to the formula, usually a row with values of first several columns in the truth table (number of columns = number of variables). A truth assignment is *satisfying* the formula if the value of the formula on these variables is T , otherwise the truth assignment is *falsifying*. A truth assignment can be encoded by a formula that is a \wedge of variables and their negations, with negated variables in places that have F in the assignment, and non-negated that have T . For example, $x = T, y = F, z = F$ is encoded as $(x \wedge \neg y \wedge \neg z)$. It is an encoding in a sense that this formula is true only on this truth assignment and nowhere else.
- Two formulas are *logically equivalent* if they have the same truth table. The most famous example of logically equivalent formulas is $\neg(p \vee q) \iff (\neg p \wedge \neg q)$ (with a dual version $\neg(p \wedge q) \iff (\neg p \vee \neg q)$) where p and q can be arbitrary (propositional, here) formulas. These pairs of logically equivalent formulas are called *DeMorgan’s law*.
- There are several other important pairs of logically equivalent formulas, called *logical identities* or *logic laws*. We will talk more about them when we talk about Boolean algebras. Here, just remember that $F \wedge p \iff p \wedge \neg p \iff F$, $F \vee p \iff T \wedge p \iff p$ and $T \vee p \iff p \vee \neg p \iff T$.
- A set of logic connectives is called *complete* if it is possible to make a formula with any truth table out of these connectives. For example, \neg, \wedge is a complete set of connectives, and so is the Sheffer’s stroke $|$ (where $p|q \iff \neg(p \wedge q)$), also called NAND for “not-and”. However, \vee, \wedge is not a complete set of connectives because it is impossible to express a truth table with 0 when all variables are 1 with them.
- An *argument* consists of several formulas called *premises* and a final formula called a *conclusion*. If we call premises $A_1 \dots A_n$ and conclusion B , then an argument is *valid* iff premises imply the conclusion, that is, $A_1 \wedge \dots \wedge A_n \rightarrow B$.

We usually write them in the following format:

Today is either Thursday or Friday
 On Thursdays I have to go to a lecture
 Today is not Friday

 \therefore I have to go to a lecture today

A valid form of argument is called *rule of inference*. The most prominent such rule is called *modus ponens*.

$p \rightarrow q$
 p —————
 $\therefore q$

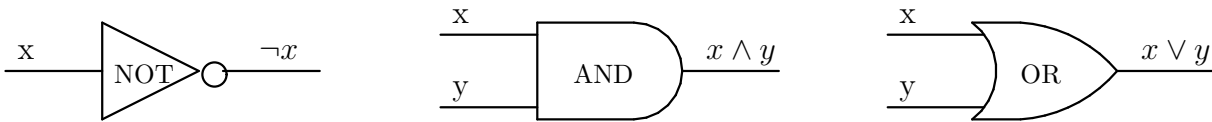


Figure 1: Types of gates in a digital circuit.

- There are several main types of proofs depending on the types of rules of inference used in the proof. The main ones are *proof by contrapositive*, *by contradiction* and *by cases*.
- There are two main normal forms for the propositional formulas. One is called *Conjunctive normal form* (CNF) and is an \wedge of \vee of either variables or their negations (here, by \wedge and \vee we mean several formulas with \wedge between each pair, as in $(\neg x \vee y \vee z) \wedge (\neg u \vee y) \wedge x$. A *literal* is a variable or its negation (x or $\neg x$, for example). A \vee of (possibly more than 2) literals is called a *clause*, for example $(\neg u \vee z \vee x)$, so a CNF is true whenever each of the clauses is true, that is, each clause has a lite). A *Disjunctive normal form* (DNF) is like CNF except the roles of \wedge and \vee are reversed. A \wedge of literals in a DNF is called a *term*. To construct a DNF and a CNF, we can start from a truth table and then for every satisfying truth assignment \vee its encoding to a DNF, and for every falsifying truth assignment \wedge the negation of its encoding to the CNF, and apply DeMorgan's law. This may result in a very large CNFs and DNFs, comparable to the size of the truth table itself. Alternatively, a CNF can be constructed from a formula by assigning a new variable v_i to every connective and rewriting the formula as a conjunction of v_1 and expressions defining v_i 's, each containing just two other variables, and then converting these expressions into small CNFs using truth tables. For example, a formula $(x \vee y) \rightarrow (\neg z)$ can be converted to a CNF by introducing variables v_1 and v_2 , then writing $v_1 \wedge (v_1 \leftrightarrow (v_2 \rightarrow \neg z)) \wedge (v_2 \leftrightarrow (x \vee y))$, then replacing each part by a CNF using truth tables.
- A *resolution proof system* is used to find a contradiction in a formula (and, similarly, to prove that a formula is a tautology by finding a contradiction in its negation). Resolution starts with a formula in a CNF form, and applies the rule “from clause $(C \vee x)$ and clause $(D \vee \neg x)$ derive clause $(C \vee D)$ until a falsity (empty clause) is reached (so in the last step one of the clauses being *resolved* contains just one variable and another clause being resolved contains just that variable's negation. Resolution can be used to check the validity of an argument by running it on the \wedge of all premises (converted, each, to a CNF) \wedge together with the negation of the conclusion.
- *Boolean functions* are functions which take as argument boolean (ie, propositional) variables and return 1 or 0 (or, the convention here is 1 instead of T, and 0 instead of F). Each Boolean function on n variables can be fully described by its truth table. A size of a truth table of a function on n variables is 2^n . Even though we often can have a smaller description of a function, vast majority of Boolean functions cannot be described by anything much smaller. Every Boolean function can be described by a CNF or DNF, using the above construction.
- *Boolean circuits* is a generalization of Boolean formulas in which we allow to reuse a part of a formula rather than writing it twice. To make a transition write Boolean formulas as trees and reuse parts that are repeating. The connectives become *circuit gates*.

It is possible to have more than 2 inputs into an AND or OR circuit, but not a NOT circuit.

It is possible to construct arithmetic circuits (e.g., for doing addition on numbers) by using a Boolean circuit to compute each bit of the answer separately.