CS2742 final exam study sheet

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 $p \to q$ (equivalent to $\neg p \vee q$), biconditional $p \leftrightarrow q$ (equivalent to $(p \to q) \wedge (q \to p)$). The order of precedence: $\neg$ strongest, $\wedge$ next, $\vee$ next, $\to$ and $\leftrightarrow$ the same, weakest.

- If $p \to q$ is an implication, then $\neg q \to \neg p$ is its *contrapositive*, $q \to p$ a *converse* and $\neg p \to \neg q$ an *inverse*. An implication is equivalent to its contrapositive, but not to converse/inverse or their negations. A negation of an implication $p \to q$ 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, $A_1 \wedge \cdots \wedge A_n \to B$. 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 (alternatively, On Friday I have to go to the lecture)

$\therefore$ 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*.

  $$\begin{array}{l} p \to q \\ p \\ \hline \therefore q \end{array}$$

- 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 for some truth assignment whenever this assignment makes each of the clauses is true, that is, each clause has a literal that evaluates to true under this assignment.. 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, 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 ($2^{\text{number of variables}}$). 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 $(x \vee y) \to (\neg z)$ can be converted to a CNF by introducing variables v_1 and v_2, then writing $v_1 \wedge (v_1 \leftrightarrow (v_2 \to \neg z)) \wedge (v_2 \leftrightarrow (x \vee y))$, 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 F (equivalently, 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.

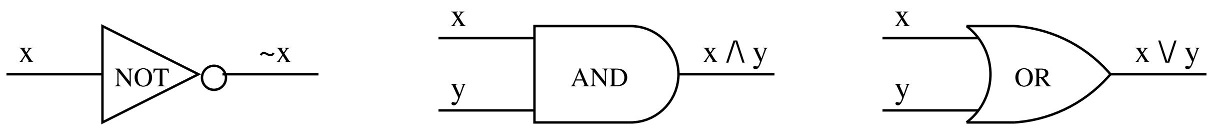


Figure 1: Types of gates in a digital circuit.

- *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.

Predicate logic:

- A *predicate* is like a propositional variable, but with *free variables*, and can be true or false depending on the value of these free variables. A *domain* of a predicate is a set from which the free variables can take their values (e.g., the domain of $Even(n)$ can be integers).

- *Quantifiers* For a predicate $P(x)$, a quantified statement "for all" ("every", "all") $\forall x P(x)$ is true iff $P(x)$ is true for every value of x from the domain (also called universe); here, $\forall$ is called a *universal quantifier.* A statement "exists" ("some", "a") $\exists x P(x)$ is true whenever $P(x)$ is true for at least one element x in the universe; $\exists$ is an existential quantifier. The word "any" means sometimes $\exists$ and sometimes $\forall$. A domain (universe) of a quantifier, sometimes written as $\exists x \in D$ and $\forall x \in D$ is the set of values from which the possible choices for x are made. If the domain of a quantifier is empty, then if the quantifier is universal then the formula is true, and if quantifier is existential, false. A *scope* of a quantifier is a part of the formula (akin to a piece of code) on which the variable under that quantifier can be used (after the quantifier symbol/inside the parentheses/until there is another quantifier over a variable with the same name). A variable is *bound* if it is under a some quantifier symbol, otherwise it is free.

- *First-order formula* A predicate is a first-order formula (possibly with free variables). A $\wedge, \vee, \neg$ of first-order formulas is a first-order formula. If a formula $A(x)$ has a free variable (that is, a variable x that occurs in some predicates but does not occur under quantifiers such as $\forall x$ or $\exists x$), then $\forall x\ A(x)$ and $\exists x\ A(x)$ are also first-order formulas.

- *Negating quantifiers.* Remember that $\neg \forall x P(x) \iff \exists x \neg P(x)$ and $\neg \exists x P(x) \iff \forall x \neg P(x)$.

- *Database queries* A *query* in a relational database is often represented as a first-order formula, where predicates correspond to the relations occurring in database (that is, a predicate is true on a tuple of values of variables if the corresponding relation contains that tuple). A query "returns" a set of values that satisfy the formula describing the query; a Boolean query, with no free variables, returns true or false. For example, a relation $StudentInfo(x, y)$ in a university database contains, say, all pairs x, y such that x is a student's name and y is the student number of student with the name x. A corresponding predicate $StudentInfo(x, y)$ will be true on all pairs x, y that are in the database. A query $\exists x StudentInfo(x, y)$ returns all valid student numbers. A query $\exists x \exists y StudentInfo(x, y)$, saying that there is at least one registered student, returns true if there is some student who is registered and false otherwise.

- *Reasoning in predicate logic* The *rule of universal instantiation* says that if some property is true of everything in the domain, then it is true for any particular object in the domain. A combination of this rule with modus ponens such as what is used in the "all men are mortal, Socrates is a man $\therefore$ Socrates is mortal" is called universal modus ponens.

- *Normal forms* In a first-order formula, it is possible to rename variables under quantifiers so that they all have different names. Then, after pushing negations into the formulas under the quantifiers, the quantifier symbols can be moved to the front of a formula (making their scope the whole formula).

- *Formulas with finite domains* If the domain of a formula is finite, it is possible to check its truth value using Resolution method. For that, the formula is converted into a propositional formula (*grounding*) by changing each $\forall x$ quantifier with a $\wedge$ of the formula on all possible values of x; an $\exists$ quantifier becomes a $\vee$. Then terms of the form $P(value)$ (e.g., $Even(5)$) are treated as propositional variables, and resolution can be used as in the propositional case.

- *Limitations of first-order logic* There are concepts that are not expressible by first-order formulas, for example, transitivity ("is there a flight from A to B with arbitrary many legs?" cannot be a database query described by a first-order formula).

Set Theory

- A *set* is a well-defined collection of objects, called elements of a set. An object x belongs to set A is denoted $x \in A$ (said "x in A" or "x is a member of A"). Usually for every set we consider a bigger "universe" from which its elements come (for example, for a set of even numbers, the universe can be all natural numbers). A set is often constructed using *set-builder notation*: $A = \{x\ inU | P(x)\}$ where U is a universe , and $P(x)$ is a predicate statement; this is read as "x in U such that $P(x)$" and denotes all elements in the universe for which $P(x)$ holds. Alternatively, for a small set, one can list its elements in curly brackets (e.g., $A = \{1, 2, 3, 4\}$.)

- A set A is a *subset* of set B, denoted$A \subseteq B$, if $\forall x(x \in A \rightarrow x \in B)$. It is a *proper* subset if $\exists x \in B$ such that $x \notin A$. Otherwise, if $\forall x(x \in A \leftrightarrow x \in B)$ two sets are equal.

- Special sets are: *empty set* $\emptyset$, defined as $\forall x(x \notin \emptyset)$. Universal set U: all potential elements under consideration at given moment. A *power set* for a given set A, denoted 2^A is the set of all subsets of A. If A has n elements, then 2^A has 2^n elements (since for every element there are two choices, either it is in, or not). A power set is always larger than the original set, even in the infinite case (use diagonalization to prove that).

- Basic set operations are a *complement* $\bar{A}$, denoting all elements in the universe that are *not* in A, then *union* $A \cup B = \{x | x \in A \text{ or } x \in B\}$, and *intersection* $A \cap B = \{x | x \in A \text{ and } x \in B\}$ and *set difference* $A - B = \{x | x \in A \text{ and } x \notin B\}$. Lastly, the Cartesian product of two sets $A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$.

- To prove that $A \subseteq B$, show that if you take an arbitrary element of A then it is always an element of B. To prove that two sets are equal, show both $A \subseteq B$ and $B \subseteq A$. You can also use set-theoretic identities.

Table 1: **Laws of boolean algebras, logic and sets**

Name	**Logic law**	**Set theory law**	**Boolean algebra law**
Double Negation	$\neg\neg p \iff p$	$\overline{\overline{A}} = A$	$\overline{\overline{x}} = x$
DeMorgan's laws	$\neg(p \vee q) \iff (\neg p \wedge \neg q)$ $\neg(p \wedge q) \iff (\neg p \vee \neg q)$	$\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$ $\overline{x \cdot y} = \overline{x} + \overline{y}$
Associativity	$(p \vee q) \vee r \iff p \vee (q \vee r)$ $(p \wedge q) \wedge r \iff p \wedge (q \wedge r)$	$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$	$(x + y) + z = x + (y + z)$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
Commutativity	$p \vee q \iff q \vee p$ $p \wedge q \iff q \wedge p$	$A \cup B = B \cup A$ $A \cap B = B \cap A$	$x + y = y + x$ $x \cdot y = y \cdot x$
Distributivity	$p \wedge (q \vee r) \iff (p \vee q) \wedge (p \vee r)$ $p \vee (q \wedge r) \iff (p \wedge q) \vee (p \wedge r)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ $x + (y \cdot z) = (x + y) \cdot (x + z)$
Idempotence	$(p \vee p) \iff p \iff (p \wedge p)$	$A \cup A = A = A \cap A$	$x + x = x = x \cdot x$
Identity	$p \vee F \iff p \iff p \wedge T$	$A \cup \emptyset = A = A \cap U$	$x + 0 = x = x \cdot 1$
Inverse	$p \vee \neg p \iff T$ $p \wedge \neg p \iff F$	$A \cup \bar{A} = U$ $A \cap \bar{A} = \emptyset$	$x + \bar{x} = 1$ $x \cdot \bar{x} = 0$
Domination	$p \vee T \iff T$ $p \wedge F \iff F$	$A \cup U = U$ $A \cap \emptyset = \emptyset$	$x + 1 = 1$ $x \cdot 0 = 0$

- A *cardinality* of a set is the number of elements in it. Two sets have the same cardinality if there is a bijection between them. If the cardinality of a set is the same as the cardinality of $\mathbb{N}$, the set is called *countable*. If it is greater, then *uncountable*.

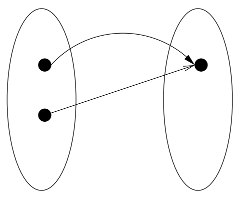
- *Principle of inclusion-exclusion*: The number of elements in $A \cup B$, $|A \cup B| = |A| + |B| - |A \cap B|$. In general, add all odd-sized intersections and subtract all even-sized intersections.

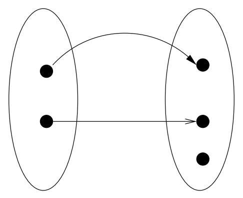
- **Boolean algebra**: A set B with three operations $+, \cdot$ and $\bar{}$, and special elements 0 and 1 such that $0 \neq 1$, and axioms of identity, complement, associativity and distributivity. Logic is a boolean algebra with F being 0, T being 1, and $\bar{}, +, \cdot$ being $\neg, \vee, \wedge$,respectively. Set theory is a boolean algebra with $\emptyset$ for 0, U for 1, and $\bar{}, \cup, \cap$ for $\bar{}, +, \cdot$. Boolean algebra is sound and complete: anything true is provable (completeness) and anything provable is true (soundness).

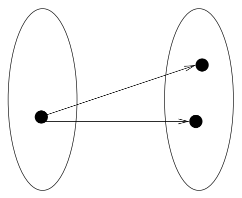
  To see that it is sound, use the fact that the axioms are true in the language of first-order logic or set theory, and the rules of inference are $x = x$, $x = y \to y = x$ and $x = y \wedge y = z \to y = z$, which preserve soundness. For completeness, show that every formula in Boolean algebra can be simplified and then extended to its "normal form" DNF obtained from its truth table.

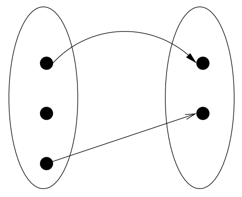
Relations and Functions

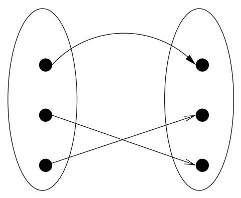
1. A k-ary **relation** R is a subset of Cartesian product of k sets $A_1 \times \cdots \times A_k$. We call elements of such R "k-tuples". A *binary* relation is a subset of a Cartesian product of two sets, so it is a set of *pairs* of elements. E.g., $R \subset \{2,3,4\} \times \{4,6,12\}$, where $R = \{(2,4),(2,6),(2,12),(3,6),(3,12),(4,4),(4,12)\}$ is a binary relation consisting of pairs of numbers such that the first number in the pair divides the second.


Not one–to–one


Not onto


Not a function


Not total


Bijection

2. Binary relation R (usually over $A \times A$) can be:
   - *reflexive*: $\forall x \in A \;\; R(x,x)$. For example, $a \leq b$ and $a = b$.
   - *symmetric*: $\forall x, y \in A \;\; R(x,y) \rightarrow R(y,x)$. For example, $a = b$, $''sibling''$.
   - *antisymmetric*: $\forall x, y \in A \;\; R(x,y) \wedge R(y,x) \rightarrow x = y$. For example, $a \leq b$, $''parent''$.
   - *transitive*: $\forall x, y, z \in A \;\; (R(x,y) \wedge R(y,z) \rightarrow R(x,z)$. For example, $a = b$, $a < b$, $a|b$, $''ancestor''$.
   - *equivalence*: if R is reflexive, symmetric and transitive. For example, $a = b$, $a \iff b$.
   - *order (total/partial)*: If R is antisymmetric, reflexive and transitive, then R is an order relation. If, additionally, $\forall x, y \in A \;\; R(x,y) \vee R(y,x)$, then the relation is a *total* order (e.g., $a \leq b$). Otherwise, it is a *partial* order (e.g., "ancestor", $a|b$.) An order relation can be represented by a Hasse diagram, which shows all connections between elements that cannot be derived by transitivity-reflexivity (e.g., "$p|n$" on $\{2, 6, 12\}$ will be depicted with just the connections 2 to 6 and 6 to 12.)
   - *transitive closure*: A transitive closure of R is a relation R^{tc} that contains, in addition to R, all x, y such that there are $k \in \mathbb{N}, v_1, \ldots, v_k \in A$ such that $x = v_1, y = v_k$, and for i such that $1 \leq i < k$, $R(v_i, v_{i+1})$. For example, an "ancestor" relation is the transitive closure of the "parent" relation.
3. A **function** f: $A \rightarrow B$ is a special type of relation $R \subseteq A \times B$ such that for any $x \in A, y, z \in B$, if $f(x) = y$ and $f(x) = z$ then $y = z$. If $A = A_1 \times \ldots \times A_k$, we say that the function is k-ary. In words, a $k+1$-ary relation is a k-ary function if for any possible value of the first k variables there is at most one value of the last variable. We also say "f is a mapping from A to B" for a function f, and call $f(x) = y$ "f maps x to y".
   - A function is *total* if there is a value $f(x) \in B$ for every x; otherwise the function is *partial.* For example, $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = x^2$ is a total function, but $f(x) = \frac{1}{x}$ is partial, because it is not defined when $x = 0$.
   - If a function is f: $A \rightarrow B$, then A is called the *domain* of the function, and B a *codomain.* The set of $\{y \in B \mid \exists x \in A, f(x) = y\}$ is called the *range* of f. For $f(x) = y$, y is called the *image* of x and x a *preimage* of y.
   - A *composition* of f: $A \rightarrow B$ and g: $B \rightarrow C$ is a function $g \circ f$: $A \rightarrow C$ such that if $f(x) = y$ and $g(y) = z$, then $(g \circ f)(x) = g(f(x)) = z$.
   - A function g: $B \rightarrow A$ is an *inverse* of f (denoted f^{-1}) if $(g \circ f)(x) = x$ for all $x \in A$.
   - A total function f is *one-to-one* if for every $y \in B$, there is at most one $x \in A$ such that $f(x) = y$. For example, the function $f(x) = x^2$ is not one-to-one when f: $\mathbb{Z} \rightarrow \mathbb{N}$ (because both $-x$ and x are mapped to the same x^2), but is one-to-one when f: $\mathbb{N} \rightarrow \mathbb{N}$.

- A total function $f\colon A \to B$ is *onto* if the range of f is all of B, that is, for every element in B there is some element in A that maps to it. For example, $f(x) = 2x$ is onto when $f\colon \mathbb{N} \to Even$, where $Even$ is the set of all even numbers, but not onto $\mathbb{N}$.
- A total function that is both one-to-one and onto is called a *bijection.*
- A function $f(x) = x$ is called the *identity* function. It has the property that $f^{-1}(x) = f(x)$. A function $f(x) = c$ for some fixed constant c (e.g., $f(x) = 3$) is called a *constant* function.

4. **Comparing set sizes**
Two sets A and B have the same cardinality if exists f that is a bijection from A to B.
If a set has the same cardinality as $\mathbb{N}$, we call it a *countable* set. If it has cardinality larger than the cardinality of $\mathbb{N}$, we call it *uncountable.* If it has k elements for some $k \in \mathbb{N}$, we call it *finite*, otherwise *infinite* (so countable and uncountable sets are infinite). E.g: $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, Even$, set of all finite strings, Java programs, or algorithms are all countable, and $\mathbb{R}, \mathbb{C}$, power set of $\mathbb{N}$, are all uncountable. E.g., to show that $\mathbb{Z}$ is countable, we prove that there is a bijection $f\colon \mathbb{Z} \to \mathbb{N}$: take $f(x) = \begin{cases} 2x & x \geq 0 \\ 1-2x & x < 0 \end{cases}$. It is one-to-one because $f(x) = f(y)$ only if $x = y$, and it is onto because for any $y \in \mathbb{N}$, if it is even then its preimage is $y/2$, if it is odd $-\frac{y-1}{2}$. Often it is easier to give instead two one-to-one functions, from the first set to the second and another from the second to the third. Also, often instead of a full description of a function it is enough to show that there is an enumeration such that every element of, say, $\mathbb{Z}$ is mapped to a distinct element of $\mathbb{N}$. To show that one finite set is smaller than another, just compare the number of elements. To show that one infinite set is smaller than another, in particular that a set is uncountable, use *diagonalization*: suppose that a there is an enumeration of elements of a set, say, $2^{\mathbb{N}}$ by elements of $\mathbb{N}$. List all elements of $2^{\mathbb{N}}$ according to that enumeration. Now, construct a new set which is not in the enumeration by making it differ from the k^{th} element of the enumeration in the k^{th} place (e.g., if the second set contains element 2, then the diagonal set will not contain the element 2, and vice versa).

Foundations of mathematics

- *Zermelo-Fraenkel set theory* The foundations of mathematics, that is, the axioms from which all the mathematics is derived are several axioms about sets called Zermelo-Fraenkel set theory (together with the axiom of choice abbreviated as ZFC). For example, numbers can be defined from sets as follows: 0 is $\emptyset$. 1 is $\{\emptyset\}$. 2 is $\{\emptyset, \{\emptyset\}\}$ and in general n is $n-1 \cup \{n-1\}$. ZFC is carefully constructed to explicitly disallow *Russell's paradox* "if a barber shaves everybody who does not shave himself, who shaves the barber?" (in set theoretic terms, if X is a set of sets A such as $A \notin A$, is $X \in X$?) This kind of reasoning is used to prove that *halting problem* of checking whether a given piece of code contains an infinite loop is not solvable (an alternative way to prove this is diagonalization).

- A *theory* is a set of statements, or, in a different view, a set of axioms from which a set of statements can be proven. Here, *axioms* are statements that are assumed in the theory (for example, $\forall x, y \;\; x+y = y+x$).

- A *model* of a theory is a description of a possible world, that is, a set of objects and interpretations of functions and relations, in which axioms are true. There can be several models for the same theory. For example, Euclidean geometry has as a model the geometry on a plane. Without the 5th postulate about parallel lines, there are other possible models such as geometry on a sphere (where parallel lines intersect).

- A theory is called *complete* if it contains (proves) every true (in every model) statement about the objects that can be described in its language (e.g., sets or natural numbers) and *sound* if every statement provable from the axioms is indeed true.
- Hilbert's program: list of problems in mathematics to solve, 1990. Includes problem 2: "prove consistency of arithmetic", as well as continuum hypothesis and a few others.
- *Gödel incompleteness theorems* say 1) for every (powerful enough and computable) theory of arithmetic efficiently computable theory that can do arithmetic ($+$ and $*$) there is a true sentence which this theory cannot prove (this is the sentence "I am not provable"): that is, if a theory is consistent, then it is not complete. 2) A (powerful enough and computable) theory of arithmetic cannot prove its own *consistency* (that is it not contradictory).
- There are examples of systems of axioms that give complete and consistent theories, but they cannot do full arithmetic: Boolean algebra or Presburger's arithmetic (which only has $+$, no $*$).

Mathematical induction, recursive definitions and algorithm analysis.

1. **Mathematical induction** Let $n_0, n \in \mathbb{N}$, and $P(n)$ is a predicate with free variable n. Then the mathematical induction principle says:

   $$(P(n_0) \wedge \forall n \geq n_0 \ (P(n) \to P(n+1))) \to \forall n \geq n_0 \ P(n)$$

   That is, to prove that a statement is true for all (sufficiently large) n, it is enough to prove that it holds for the smallest $n = n_0$ (*base case*) and prove that if it holds for some arbitrary $n > n_0$ (*induction hypothesis*) then it also holds for the next value of n, $n+1$ (*induction step*).

   A *strong induction* is a variant of induction in which instead of assuming that the statement holds for just one value of n we assume it holds for several: $P(k) \wedge P(k+1) \wedge \cdots \wedge P(n) \to P(n+1)$ instead of just $P(n) \to P(n+1)$. In particular, in *complete induction* $k = n_0$, so we are assuming that the statement holds for *all* values smaller than $n+1$.

   A *well-ordering principle* states that every set of natural numbers has the smallest element. It is used to prove statements by counterexample: e.g., "define set of elements for which $P(n)$ does not hold. Take the smallest such n. Show that it is either not the smallest, or $P(n)$ holds for it".

   These three principles, Induction, Strong Induction and Well-ordering are equivalent. If you can prove a statement by one of them, you can prove it by the others.

   The following is the structure of an induction proof.

   (a) $P(n)$. State which assertion $P(n)$ you are proving by induction. E.g., $P(n)$: $2^n < n!$.

   (b) Base case: Prove $P(n_0)$ (usually just put n_0 in the expression and check that it works). E.g., $P(4)$: $2^4 < 4!$ holds because $2^4 = 16$ and $4! = 24$ and $16 < 24$.

   (c) Induction hypothesis: "assume $P(n)$ for some $n > n_0$". I like to rewrite the statement for $P(n)$ at this point, just to see what I am using. For example, "Assume $2^n < n!$".

(d) Induction step: prove $P(n+1)$ under assumption that $P(n)$ holds. This is where all the work is. Start by writing $P(n+1)$ (for example, $2^{n+1} < (n+1)!$. Then try to make one side of the expression to "look like" (one side of) the induction hypothesis, maybe + some stuff and/or times some other stuff. For example, $2^{n+1} = 2 \cdot \mathbf{2^n}$, which is 2^n times additional 2. The next step is either to substitute the right side of induction hypothesis in the resulting expression with the left side (e.g., 2^n in $2 \cdot 2^n$ with $n!$, giving $2 \cdot n!$, or just apply the induction hypothesis assumption to prove the final result You might need to do some manipulations with the resulting expression to get what you want, but applying the induction hypothesis should be the main part of the proof of the induction step.

2. A *recursive definition* (of a set) consists of 1) The base of recursion: "these several elements are in the set". 2) The recursion: "these rules are used to get new elements". 3) A restriction: "nothing extraneous is in the set".

3. A *Structural induction* is used to prove properties about recursively defined sets. The base case of the structural induction is to prove that $P(x)$ holds for the elements in the base, and the induction steps proves that if the property holds for some elements in the set, then it holds for all elements obtained using the rules in the recursion.

4. Recursive definitions of functions are defined similar to sets: define a function on 0 or 1 (or several), and then give a rule for constructing new values from smaller ones. Some recursive definitions do not give a well-defined function (e.g., $G(n) = G(n/2)$ if n is even and $G(3n+1)$ if N is odd is not well defined). Some functions are well-defined but grow extremely fast: Ackermann function defined as $\forall m, n > 0, A(0, n) = n + 1, A(m, 0) = A(m-1, 1), A(m, n) = A(m-1, A(m, n-1))$

5. To compare grows rate of the functions, use $O()$-notation. $f(n) \in O(g(n))$ if $\exists n_0, c > 0$ such that $\forall n \geq n_0 \; f(n) \leq cg(n)$. In algorithmic terms, if $f(n)$ and $g(n)$ are running times of two algorithms for the same problem, $f(n)$ works faster on large inputs.

6. **Algorithm correctness**

   (a) Preconditions and postconditions for a piece of code state, respectively, assumptions about input/values of the variables before this code is executed and the result after. If the code is correct, then preconditions + code imply postconditions.

   (b) A *Loop invariant* is used to prove correctness of a loop. This is a statement implied by the precondition of the loop, true on every iteration of the loop (with loop guard variables as a parameter) and after the loop finishes implies the postcondition of the loop. A *guard* condition is a check whether to exist the loop or do another iteration. To prove total correctness of a program, need to prove that eventually the guard becomes false and the loop exists (however, it is not always possible to prove it given somebody's code), otherwise, the proof is of partial correctness.

   (c) Correctness of recursive programs is proven by (strong) induction, with base case being the tail recursion call, and the induction step assumes that the calls returned the correct values and proves that the current iteration returns a correct value.