Exam preparation cheetzheet: *Stuff from*

MNF130V2020

May 20, 2020

1 Chapter 1

Propositional logic:. Logical $\wedge, \vee, \oplus$ are trivial.
Conditional statements (implication): $p \rightarrow q$, if p then $q \equiv p$ only if $q \equiv p$ is a sufficient condition for q.
In other words, q is a necessary condition for p. $p \rightarrow q$ is false then p is true and q is false and otherwise true. $\neg(p \rightarrow q) \equiv p \wedge (\neg q)$, $p \rightarrow q$ is equivalent to its contrapositive $\neg q \rightarrow \neg p$, but **not** to its **converse** $q \rightarrow p$ **or** its inverse $\neg p \rightarrow \neg q$.
Biconditional statements: $p \leftrightarrow q$ or expanded to $(p \rightarrow q) \wedge (q \rightarrow p)$.
De Morgan: $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$; $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$ Propositional logic can be represented by gates, creating combinational circuits which can represent **any** logical expression.
Quantifiers:

$\forall x(P(x) \rightarrow Q(x)) \equiv$ *for all x, if P(x) then Q(x)*
$\exists x(P(x) \wedge Q(x)) \equiv$ *there exists an x such that P(x) and Q(x)*

P(x),Q(x) are propositional functions and there is always a **domain** or **universe of discourse**, either implicit or explicitly stated,over which the variable ranges.
Negations of quantified propositions: $\neg\forall x P(x) \equiv \exists x \neg P(x); \neg\exists P(x) \equiv \forall x \neg P(x)$.
Theorem: A proposition that can be proved; **lemma:** a simple theorem, commonly used as part of a greater picture to prove other theorems; **proof:** A demonstration that a proposition is true, **collorary:** A proposition that can be proved as a consequence of a theorem that has just been proved. A collorary can be seen as “Side effects“ of the prooved theorem.
A **valid** argument is an argument using correct rules of inference based on tautologies (something that will always give the **true** conclusion in **any** given scenario. I. E. a tautology is something that is always true for all possible combinations.)
An **invalid** argument can be referred to as a **fallacy**, such as affirming the conclusion, denying the hypothesis, begging the question or circular reasoning. They can lead to false conclusions.
Some rules of inference:

- $[p \wedge (p \rightarrow q)]$ Modus Ponens
- $[\neg q \wedge (p \rightarrow q)]$ Modus Tollens
- $[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$ Hypothetical syllogism (Transitivity)
- $[(p \vee q) \wedge (\neg p)] \rightarrow q$ Disjunctive syllogism
- $\{P(a) \wedge \forall x[P(x) \rightarrow Q(x)]\} \rightarrow Q(a)$ Universal modus ponens
- $\{\neg Q(a) \wedge \forall x[P(x) \rightarrow Q(x)]\} \rightarrow \neg P(a)$ Universal modus tollens
- $(\forall x P(x)) \rightarrow P(c)$ Universal instantiation
- $(P(c) arbitrary\ c) \rightarrow \forall x P(x)$ Universal generalization
- $(\exists x P(x)) \rightarrow (P(c)\ for\ some\ c)$ Existential instantiation
- $(P(c)\ for\ some\ element\ c) \rightarrow \exists x P(x)$ Existential generalization.

a Proofs

Trivial proof: A proof that $p \to q$ just shows that q is true witout using the hypothesis p.
Vacuous proof: A proof of $p \to q$ that just shows that the hypothesis p is false.
Direct proof: A proof of $p \to q$ that shows that the assumption of the hypothesis p implies the conclusion of q.
Proof by contraposition: A proof of $p \to q$ that shows that the assumption of the negation of the conclusion q implies the negation of the hypothesis p (in other words, proof of contrapositive).
Proof by contradiction: A proof of p that shows that the assumption of the negation of p leads to a contradiction.
Proof by cases: A proof of $(p_1 \vee p_2 \vee p_3 ... p_n) \to q$ that shows that each conditional statement $p_i \to q$ is true.
Statements of the form $p \leftrightarrow q$ require that both $p \to q$ and $q \to p$ be proved. It is sometimes necessary to give the two separate proof (usually a direct proof or a proof by contraposition); other times a string of equivalences can be constructed starting with p and ending with $q : p \leftrightarrow p_1 \leftrightarrow p_2 ... \leftrightarrow p_n \leftrightarrow q$.
To give a **constructive proof** of $\exists x P(x)$ is to show how to find an element x that makes $P(x)$ true. **Non-constructive existence proofs** are also possible, often using **proof by contradiction**.
One can **disprove** a universally quantified proposition $\forall x P(x)$ simply by giving a **counter example**, i.e. an object x such that $P(x)$ is **false**. One can, however, not proove it with such an example.
Fermat's last theorem: There are no positive integer solutions of $x^n + y^n = z^n \ if \ n > 2$.

An integer is **even** if it can be written as $2k$ for some integer k; an integer is **odd** if it can be written as $2k+1$ for some integer k. Every number is even or odd but not both. A number is **rational**, if it can be written as p/q with p being an integer and q strictly a non-zero integer.

2 Chapter 2

a Sets

Empty set: A set with no elements, commonly denoted as $\emptyset$. Do not confuse this with the set only containing the empty set. The difference is that the empty itself is empty, whereas the set containing the empty set has a single element.
Subset: $A \subseteq B \equiv \forall x(x \in A \rightarrow x \in B)$, whereas a proper subset is $A \subset B \equiv (A \subseteq B) \wedge (A \neq B)$, in other words, B has at least one element different from the set A.
Equality of sets: $A = B \equiv (A \subseteq B \wedge B \subseteq A) \equiv \forall x(x \in A \leftrightarrow x \in B)$.
Power set: $\mathcal{P}(A) = \{B|B \subseteq A\}$, the set of all subsets of A. A set with n elements has 2^n subsets.
Cardinality: $|S|$, the number of elements in S.
Some specific sets in regards to cardinality: $\mathbb{R}$ is the set of real numbers, represented by either finite or infinite decimals;
$\mathbb{N}$ is the set of all natural numbers (eg. $\{0,1,2,3,4,5...\}$), $\mathbb{Z}$ is the set of integers $\{...-2,-1,0,1,2,...\}$ and can also be denoted with only the positive or negative subset. $\mathbb{Q}$ is the set of rational numbers, where $\{p/q|p,q \in \mathbb{Z} \wedge q \neq 0\}$, $\mathbb{Q}^+$ is the set of positive rational numbers and a subset of $\mathbb{Q}$.
Set operations: $A \times B = \{(a,b)|a \in A \wedge b \in B\}$ (**Cartesian Product**); $\overline{A}$ is the set of elements in the universe which are **not** in A (**complement**); $A \cap B = \{x|x \in A \wedge x \in B\}$ (**intersection**); $A \cup B = \{x|x \in A \vee x \in B\}$ (**union**); $A - B = A \cap \overline{B}$ (**difference**); $A \oplus B = (A-B) \cup (B-A)$, (**symmetric difference/xor**)
Inclusion-exclusion (simple case): $|A \cup B| = |A| + |B| - |A \cap B|$
De Morgan's laws for sets: $\overline{A \cap B} = \overline{A} \cup \overline{B}$; $\overline{A \cup B} = \overline{A} \cap \overline{B}$
A **function** f from A (**the domain**) to B (**the co-domain**) is an assignment of a unique element of B to each element of A. Write $f : A \rightarrow B$. Write $f(a) = b$ if b is assigned to a. **Range** of f is $\{f(a)|a \in A\}$; f is **onto/surjective** $\equiv$ range (f) = B; f is **one-to-one/injective** $\equiv \forall a_1 \forall a_2[f(a_1) = f(a_2) \rightarrow a_1 = a_2]$
If f is one-to-one **and** onto, it is **bijective** and the **inverse** function $f^{-1} : B \rightarrow A$ is defined by $f^{-1}(y) = x \equiv f(x) = y$.
If $f : B \rightarrow C$ and $g : A \rightarrow B$, then the **composition** $f \circ g$ is the function from A to C defined by $f \circ g(x) = f(g(x))$.
Rounding functions: $\lfloor x \rfloor$ is the largest integer less than or equal to x **floor function**; $\lceil x \rceil$ is the smallest integer greater than or equal to x **the ceiling function**.
Summation notation:

$$\sum_{n=1}^{n} a_i = a_1 + a_2 + a_3 + ... + a_n$$

Sum of first n positive integers:

$$\sum_{j=1}^{n} j = 1 + 2 + ... + n = \frac{n(n+1)}{2}$$

Sum of squares of first n positive integers:

$$\sum_{j=1}^{n} j^2 = 1^2 + 2^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Sum of geometric progression: (I don't think we did this in the course)

Two sets are said to have the **same cardinality** if there is a **bijection** between them. We can say that $|A| \leq |B|$ if there is a one-to-one function from A to B.
A set is *countable* if it is finite or there is a **bijection** from the positive integers to the set. **In other words**, if the elements of the set can be listed (e.g. $a_1, a_2, ...$). Sets of the latter type are called $countablyinfinite$ and the **cardinality of these sets are denoted with** $\aleph_0$. The empty set, the integers and the rational numbers **are countable**. The union of a countable number of countable sets is countable.
Schroder-Bernstein theorem: If $|A| \leq |B|$ and $|B| \leq |A|$ then it must be that $|A| = |B|$. This can be explained as if there is a one-to-one function from A to B and a one-to-one function from B to A, then there is a one-to-one and onto function from A to B.
Matrix Multiplication: The $(i, j)^{th}$ entry of **AB** is $\sum_{t=1}^{k} a_{it} b_{tj}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, where **A** is an $m \times k$ matrix and **B** is a $k \times n$ matrix.
Identity matrix I_n with 1's on the main diagonal and 0's elsewhere is the multiplicative identity.
Cardinality arguments can be used to show that some functions are **uncomputable**.
Matrix addition (+), Boolean meet ($\wedge$) and join ($\vee$) are done entry-wise; Boolean matrix product ($\odot$) is like matrix multiplication using boolean operators.
Transpose: $\mathbf{A}^t$ is the matrix whose $(i, j)^{th}$ entry is a_{ij} (the $(j, i)^{th}$ entry of **A**);
A is **symmetric** if $A^t = A$;

3 Chapter 3

Algorithm are commonly expressed in **pseudo-code** when not directly implemented in a domain specific area.
Keywords for algorithms: {input, output, definiteness, correctness, finiteness, effectiveness, generality}.
Greedy algorithms: Will examine and pick the best choice at a given step. Not always the best.
Brute forcing: Specifically in discrete mathematics, this referres to examining the entire space of solutions and then determine the best one (very inefficient, sometimes necessary). Not explained in this course: **dynamic programming, probabilistic algorithms, divide-and-conquer**.
Halting problem: The unsolvable computing problem whether a program will halt given input. (Alan Turing for reference...)
Big-O: Half of inf102 is just this:
$f(x) = O(g(x))$ means $\exists C \exists k \forall x(x > k \rightarrow |f(x)| \leq C|g(x)|)$. Big-O of a sum is the largest (fastest growing) of the functions in the sum. Big-O of a product is the product of the big-O's of the factors. If f is $O(g)$, the g is $\Omega(f)$ "big-omega". If f is both big-O and big-Omega of g, then f is $\theta(g)$ "big-theta".
Little-O: We say that $f(x)$ is $o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$.
Powers grow faster than logs: $(\log n)^c$ is $O(x^d)$ but not the other way around, where c and d are positive numbers.
If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$ and $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.
$\log n!$ is $O(n \log n)$.
Time complexity: Binary search = $O(\log n)$ (cut half of possibilites at each step), linear search $O(n)$ (all input is examined exactly once), both have **space complexity (in terms of computer memory) O(1)** without taking the input into account. Bubble sort and insertion sort have $O(n^2)$.
Matrix multiplication has standard algorithm time complexity of $O(m_1 m_2 m_3)$ if the matrices have dimensions $m_1 \times m_2$ and $m_2 \times m_3$.
Efficient algorithms can reduce the complexity of multiplying two $n \times x$ matrices from $O(n^3)$ to $O(n^{\sqrt{7}})$ Important complexity classes include polynomials n^b, exponential (b^n for $b > 1$) and factorial ($n!$).
A problem that can be solved by an algorithm with polynomial worst-case time complexity is called **tractable**; otherwise **intractable**.
P=NP problem: The class **P** is the class of tractable problems. The class **NP** consists of the problems for which it is possible to check solutions (**not FIND solutions**) in polynomial time. This means that $P \subseteq NP$ yet the **P=NP** problem is unsolved because it has not been shown whether **P=NP**.

4 Chapter 4

Divisibility: $a|b$ means $a \neq 0 \land \exists c(a \cdot c = b)$ (a is a **divisor** or **factor** of b such that b is a multiple of a).
Base b **representiations:** $(a_{n-1}a_{n-2}...a_2a_1a_0)_b = a_{n-1}b^{n-1} + ... + a_2b^2 + a_1b + a_0$.
To convert from base 10 to base b, continually divide by b and record remainders as $a_0, a_1, a_2, ...$ ($b = 8$ is **octal**, $b = 16$ is **hexadecimal**, using A-F for 10-15). Convert from binary to octal by grouping bits by **threes**, from the right, to hexadecimal by grouping by fours; because $2^3 = 8$ and $2^4 = 16$.
Addition: of two **binary numerals** each of n bits ($(a_{n-1}a_{n-2}...a_2a_1a_0)_2$) requires $O(n)$ bit operations.
Multiplication: requires $O(n^2)$ bit operations if done naively, $O(n^{1.585})$ steps by more sophisticated algorithms.
Division "algorithm":

$$\forall a \forall d > 0 \exists q \exists r(a = dq + r \land 0 \leq r < d)$$

where q is the quotient and r is the remainder; we write a **mod** d for the remainder.
Example of the division "algorithm":

$$-18 = 5 \cdot (-4) + 2 \rightarrow -15 \textbf{ mod } 5 = 2$$

Congruent modulo m**:** $a \equiv b(\ mod\ m) \ \leftrightarrow\ m|a-b \ \leftrightarrow\ a\ mod\ m = b\ mod\ m$
One can do arithmetic in $\mathbb{Z}_m = \{0, 1, ..., m-1\}$ by working modulo m. There are fast algorithms for computing $b^n\ mod\ m$, based on successive squaring.
Integer $n > 1$ is said to be **prime** $\leftrightarrow$ its only factors are 1 and itself; otherwise it is referred to as a **composite**.
There are infinitely many **primes**, but it is not known whether there are infinitely many twin primes (**primes that differ by 2**), or whether every even positive integer greater than 2 is the sum of two primes (**Goldbach's conjecture**) or whether there are infinitely many **Mersenne primes** $\rightarrow$ **primes of the form** $2^p - 1$.
Naive test for primeness and test for prime factorization: To find prime factorization of n, successively divide it by all primes less than $\sqrt{n}$; if none is found, then **n is prime**. If a prime factor p is found, then continue the process to find the prime factorization of the remaining factor, namely n/p; this time the trial divisions can start with p. Continue until a prime factor remains.
Prime number theorem states that there are approximately $n/\ln(n)$ primes less than or equal to n.
Fundamental theorem of arithmetic: Every integer greater than 1 can be written as a product of one or more primes, and the product is unique except for the order of the factors. (A proof based on fact that if a prime divides a product of integers, then it divides at least one of those integers.)
Euclidean algorithm for greatest common divisor: $gcd(x, y) = gcd(y, x \textbf{ mod } y)$ if $y \neq 0$; $gcd(x, 0) = x$.
Using extended Euclidean algorithm or working backwards, one can find **Bezout coefficients** and write $gcd(a, b) = sa + tb$.
Two integers are **relatively prime** if their greatest common divisor (gcd) is 1. The integers $a_1, a_2, ..., a_n$ are **pairwise relatively prime** $\leftrightarrow$ $gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.
Chinese reminder theorem: If $m_1, m_2, ...m_n$ are pairwise relatively prime, then the system $\forall i(x \equiv a_i(\ mod\ m_i))$ has unique solution modulo $m_1m_2...m_n$. An example of application of this: Handling very large integers on a computer.
Fermat's little theorem: $a^{p-1} \equiv 1(\ mod\ p)$ if p prime and does not divide a. The converse is not true; for example $2^{340} \equiv 1(\ mod\ 341)$ so $341 (= 11 \cdot 31)$ is referred to as a **pseudo prime**;
If a and b are positive integers, then there exist integers s and t such that $as + bt = gcd(a, b)$ **linear combination**.
This theorem allows one to compute the **multiplicative inverse** $\overline{a}$ of a *modulo* b (i.e $\overline{a}a \equiv 1(\ mod\ b)$) as long as a and b are relatively prime, which enables one to solve **linear congruences** $ax \equiv c(\ mod\ b)$.
A primitive root modulo a prime p is an integer r in $\mathbb{Z}_p$ such that every nonzero element of $\mathbb{Z}_p$ is a power of r.
Discrete logarithms: $\log_r a = e$ modulo p if r^e **mod** $p = a$ and $1 \leq e \leq p-1$
A common **hashing function:** $h(k) = k$ **mod** m, where k is the key.
Check digits for error-correcting codes like UPCs, involve modular arithmetic (??????????)
Pseudorandom numbers: can be generated by the **linear congruential method:** $x_{n+1} = (ax_n + c)$ **mod** m, where

x_0 is arbitrarily chosen **seed**. Then $\{x_n/m\}$ will be rather randomly distributed numbers between 0 and 1.
Shift cipher: $f(p) = (p+k)$ **mod** $26[A \leftrightarrow 0, B \leftrightarrow 1, ...]$. Caeser cipher used $k = 3$.
Affine cipher: Uses $f(p) = (ap+b)$ **mod** 26 with $gcd(a, 26) = 1$.
RSA public key encryption system: An integer M representing the plaintext is translated into an integer C representing the ciphertext using the function $C = M^e$ **mod** b, where n is a public numbers that is the product of two large (100-digit or so) primes and e is a public number *relatively* prime to $(p-1)(q-1)$; the primes p and q are kept secret. Decryption is accomplished via $M = C^d$ **mod** n, where d is an inverse of e modulo $(p-1)(q-1)$. It is infeasible to compute d without knowing p and q, which are infeasible to compute from n.
Similar methods can be used for **key exchange protocols**, **digital signatures**, **signing stuff in general**.

5 Chapter 5

The well-ordering property: Every non-empty set of nonnegative integers has a "least element".
The principle of mathematical induction: Let $P(n)$ be a propositional function in which the domain (the universe of discourse) is the set of positive integers. Then if one can show that $P(1)$ is true (through **Base case/Base step**) and that for every positive integer k, the conditional statement $P(k) \to P(k+1)$ is true **(inductive step)**, then one has proved that $\forall n P(n)$. The hypothesis $P(k)$ in a proof of the inductive step is called the **inductive hypothesis**.
More generally, the indcution can start at any integer, and there could potentially be several base cases.
Strong induction: Let $P(n)$ be a propositional function in which the domain (again, **universe of discourse**) is the set of positive integers. Then if one can show that $P(1)$ is true, and that for every positive integer k the conditional statement $[P(1) \wedge P(2) \wedge ... \wedge P(k)] \to P(k+1)$ is true **(inductive step)**, then one has proved $\forall n P(n)$. The hypothesis $\forall j \leq k P(j)$ in a proof of the inductive step is called the (**(strong) inductive hypothesis**). Again, the induction can start at any integer, and there can be several base cases.
Inductive/Recursive definitions (functions): Is a definition of a function f with the set of nonnegative integers as its domain: Specification of $f(0)$, together with, for each $n > 0$, a rule for finding $f(n)$ from values of $f(k)$ for $k < n$.
Example: $0! = 1$ and $(n+1)! = (n+1) \cdot n!$ **(factorial function)**
Inductive/Recusive definitions (sets): Definition of a set S: A rule specifying one or more particular elements of

S, together with a rule for obtaining more elements of S from those already in it. It is understood that S consists precisely of those elements that can be obtained by applying these two rules.
Structural induction: can be used to prove facts about recursively defined objects.
Fibonacci numbers: $f_0, f_1, f_2, ... : f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$ for all $n \geq 2$.
Lame's theorem: The number of divisions used by the Euclidean algorithm to find $gcd(a,b)$ is $O(\log b)$.
An algorithm is **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input. It is **iterative** if it is based on the repeated use of operations in a loop.
There is an efficient recursive algorithm for computing **modular powers (**b^n **mod** m**)**, based on computing $b^{\lfloor \frac{n}{2} \rfloor}$ **mod** m.
Merge sort: is an efficient recursive algorithm for sorting a list: break the list into two parts, recursively sort each half, and then merge them together in order. It has $O(n \log n)$ time complexity in **all** cases.
A program segment S is **partially correct** with respect to **initial assertion** p and **final assertion** q, written $p\{S\}q$, if whenever p is true for the input values of S and S terminates, q is true for the output values of S.
A **loop invariant** for **while** *condition* S is an assertion p that remains true each time S is executed in the loop; i.e. $(p \wedge \; condition)\{S\}p$. If p is true before the program segment is executed, then p and $\neg condition$ are true after it terminates (if it terminates at all). In symbols, $p\{\textbf{while}\ condition\ S\}(\neg condition\ \wedge p)$.