CS103 Oracle

November 26, 2020

Contents

1 Numbers **1**

2 Sets **1**

3 Logic **2**

4 Binary Relations **3**

5 Function **4**

6 Graphs **4**

7 Induction **6**

8 Regular Languages **6**

9 Irregular Languages **7**

10 Context Free Languages **8**

11 Computability Theory **8**

12 Complexity Theory **9**

1 Numbers

Definition 1.1 (Odd and Even)**.** An integer n is *even* if there is some integer k such that $n = 2k$. An integer n is *odd* if there is some integer k such that $n = 2k + 1$.

Theorem 1.1 (Square of even number is even)**.** *If n is an even integer, then n^2 is also even.*

Theorem 1.2 (Square root of 2 is irrational)**.** $\sqrt{2}$ *is irrational.*

2 Sets

Definition 2.1 (Empty Set)**.** The empty set, denoted $\emptyset$ or $\{\}$ is the set containing no elements.

Definition 2.2 (Subset)**.** A set A is a *subset* of another set B if for all objects x, if $x \in A$, then $x \in B$, that is all elements of A are also elements of B.

Definition 2.3 (Set equality)**.** A set A is *equal* to another set B if $A \subseteq B$ and $B \subseteq A$.

Theorem 2.1 (Transitivity of Subset)**.** *If $A \subseteq B$, and $B \subseteq C$, then $A \subseteq C$*

Definition 2.4 (Union)**.** $A \cup B = \{x : x \in A \text{ or } x \in B \text{ or both }\}$

Definition 2.5 (Intersection)**.** $A \cap B = \{x : x \in A \text{ and } x \in B\}$

Definition 2.6 (Difference)**.** $A - B = \{x : x \in A \text{ and } x \notin B\}$

Definition 2.7 (Symmetric difference)**.** $A \Delta B = \{x : x \in A \text{ and } x \notin B \text{ or } x \in B \text{ and } x \notin A\}$

Theorem 2.2 (Cantor's theorem)**.** *If S is a set, then $|S| < |\wp(S)|$. That is, the cardinality of any set is strictly smaller than the cardinality of its powerset.*

Definition 2.8 (Disjoint)**.** Two sets A, B are *disjoint* if their intersection is the emptyset.

Definition 2.9 (Partition)**.** A *partition* of a set is a way of splitting it into disjoint, non-empty subsets so that every element belongs to exactly one subset.

3 Logic

Definition 3.1 (Vacuous Truths)**.** A statement of the form $\forall x \in A.(P(x))$ is called vacuously true if $A = \emptyset$. An implication $p \implies q$ is called vacuously true if p is false.

Definition 3.2 (Negation)**.** If φ is a proposition, its *negation* $\neg\varphi$ is a proposition that is true whenever φ is false, and false whenever φ is true. TODO link guide to negations.

Definition 3.3 (Contrapositive)**.** The *contrapositive* of an implication $p \implies q$ is the equivalent statement $\neg q \implies \neg p$.

Definition 3.4 (Proof by contrapositive)**.** One way to prove an implication $p \implies q$ is to prove the contrapositive statement: $\neg q \implies \neg p$.

Definition 3.5 (Proof by contradiction)**.** To prove a proposition P. Assume its negation, $\neg P$, and find some sort of contradiction/inconsistency.

Definition 3.6 (Propositional logic)**.** TODO link.

Theorem 3.1 (De Morgan's Laws)**.** *a.)* $\neg(a \wedge b) = \neg a \vee \neg b$

b.) $\neg(a \vee b) = \neg a \wedge \neg b$

Definition 3.7 (Aristotelian Forms)**.** a.) "All A's are B's": $\forall x.(A(x) \implies B(x))$

b.) "Some A's are B's": $\exists x.(A(x) \wedge B(x))$

c.) "No A's are B's": $\exists x.(A(x) \implies \neg B(x))$

d.) "Some A's aren't B's": $\exists x.(A(x) \wedge \neg B(x))$

4 Binary Relations

Definition 4.1 (Binary Relation)**.** A *binary relation* over a set A is a predicate R that can be applied to pairs of elements drawn from A. If R holds for two elements $a, b \in A$, we write aRb. If R does not hold we write $a \not R b$

Definition 4.2 (Reflexivity)**.** A binary relation R over a set A is *reflexive* if everything relates to itself.

$$\forall a \in A.(aRa).$$

Definition 4.3 (Irreflexivity)**.** A binary relation R over a set A is *irreflexive* if nothing relates to itself.

$$\forall a \in A.(a \not R a).$$

Definition 4.4 (Symmetric)**.** A binary relation R over a set A is *symmetric* if

$$\forall a, b \in A.(aRb \implies bRa).$$

Definition 4.5 (Asymmetric)**.** A binary relation R over a set A is *asymmetric* if

$$\forall a, b \in A.(aRb \implies b \not R a).$$

Definition 4.6 (Transitive)**.** A binary relation R over a set A is *transitive* if

$$\forall a, b, c \in A.((aRb \wedge bRc) \implies aRc).$$

Definition 4.7 (Cyclic)**.** A binary relation R over a set A is *cyclic* if

$$\forall a, b, c \in A.((aRb \wedge bRc) \implies cRa).$$

Definition 4.8 (Equivalence relation)**.** A relation R over a set A is called an *equivalence relation* if it is reflexive, symmetric and transitive.

Definition 4.9 (Equivalence class)**.** If R is an equivalence relation over a set A, and $a \in A$. The *equivalence class* of a, denoted by $[a]_R$ is the set of elements that relate to a, formally,

$$[a]_R = \{b \in A : aRb\}.$$

Theorem 4.1 (Fundamental theorem of equivalence classes)**.** *Let R be an equivalence relation over a set A. Then each element $a \in A$ is in exactly one equivalence class of R. In other words, the equivalence classes of R partition A.*

Definition 4.10 (Strict order)**.** A relation R over a set A is a *strict order* if it is irreflexive, asymmetric, and transitive.

5 Function

Definition 5.1 (Function)**.** $f : A \to B$ is called a *function* from A (domain) to B (codomain) if it obeys the domain/codomain rule

$$\forall x \in A.(f(x) \in B),$$

and is deterministic

$$\forall x, y \in A.(x = y \implies f(x) = f(y)).$$

Definition 5.2 (Function composition)**.** If $f : A \to B$ and $g : B \to C$ are functions, the *composition*, $g \circ f : A \to C$, is a function such that for all $x \in A.(g \circ f(x) = g(f(x)))$

Definition 5.3 (Injective)**.** A function $f : A \to B$ is *injective* if

$$\forall x, y \in A.(f(x) = f(y) \implies x = y),$$

or equivalently,

$$\forall x, y \in A.(x \neq y \implies f(x) \neq f(y)).$$

Definition 5.4 (Surjective)**.** A function $f : A \to B$ is *surjective* if

$$\forall b \in B.\exists a \in A.(f(a) = b).$$

Definition 5.5 (Bijective)**.** A function $f : A \to B$ is *bijective* if it is both injective and surjective.

Definition 5.6 (Inverse)**.** The *inverse* of a function $f : A \to B$ is a function $f^{-1}B \to A$ that satisfies the following:

$$\forall a \in A.(f^{-1}(f(a)) = a),$$

and

$$\forall b \in B.(f(f^{-1}(b)) = b).$$

Definition 5.7 (Cardinality)**.** If X and Y are sets, $|X| = |Y|$ if there exists a bijection $f : X \to Y$, and $|X| \neq |Y|$ if every function from X to Y is not a bijection.

6 Graphs

Definition 6.1 (Graph)**.** A *graph* $G = (V, E)$ is an tuple of two sets V, and E. Where V is a set of vertices and E is a set of edges which are unordered pairs of vertices. A directed graph is the same but edges are ordered pairs instead of unordered. In this class we mainly care about undirected graphs. Self loops are not allowed in this class (no vertex can have an edge to itself).

Definition 6.2 (Adjacent)**.** If we have a graph $G = (V, E)$, two vertices u, v are *adjacent* if $\{u, v\} \in E$, that is, there is an edge between u and v.

Definition 6.3 (Degree)**.** The *degree* of a vertex v is the number of nodes that v is adjacent to.

Definition 6.4 (Path)**.** A *path* is a sequence of one or more nodes $v_1, ..., v_n$ such that any two consecutive nodes are adjacent. The length of a path is the number of edges in the path.

Definition 6.5 (Cycle)**.** A *cycle* is a path that starts and ends at the same node.

Definition 6.6 (Simple path)**.** A *simple path* is a path that does not repeat any nodes or edges.

Definition 6.7 (Simple cycle)**.** A *simple cycle* is a cycle that does not repeat any vertices except the first/last one.

Definition 6.8 (Connected)**.** Two nodes u, v in a graph are *connected* if there is a path from one to the other. A graph G is called *connected* if all pairs of nodes are connected.

Definition 6.9 (Connected component)**.** The connected component of a vertex v, denoted $[v] = \{x \in V : v \text{ is connected to } x\}$.

Theorem 6.1 (Connected is an equivalence relation)**.** *The connectivity relation on V is an equivalence relation, and the equivalence classes of this relation are connected components.*

Definition 6.10 (Planar graph)**.** A graph is called *planar* if there is some way to draw it in 2D plane without any edges crossing.

Definition 6.11 (k-coloring)**.** A *k-coloring* of a graph is a way to color each of the vertices such that no adjacent vertices are the same color. Formally, a k-coloring is a function $f : V \to \{1, 2, ..., k\}$ such that $\forall u, v.(\{u, v\} \in E \implies f(u) \neq f(v))$.

Definition 6.12 (k-colorable)**.** A graph is called *k-colorable* if a k-coloring exists.

Definition 6.13 (Chromatic number)**.** The smallest k for a graph G is k-colorable is its *chromatic number*.

Theorem 6.2 (Pigeonhole principle)**.** *If there are m objects distributed into n bins with $m > n$, then at least one bin will contain at least two objects.*

Definition 6.14. A *tree* is a connected graph with no simple cycles. They are maximally acyclic in that adding any edge creates a cycle, and minimally connected in that remove any edge makes the graph disconnected.

Theorem 6.3 (Two nodes of same degree)**.** *In any graph with at least 2 nodes, there are at least nodes of the same degree.*

Theorem 6.4 (Generalized pigeonhole principle)**.** *If you distribute m objects into n bins, then*

- *Some bin will have at least $\lceil m/n \rceil$*
- *Some bin will have at most $\lfloor m/n \rfloor$*

Theorem 6.5 (Friends and strangers)**.** *In any group of* 6 *people where every pair of people are either friends or strangers, there must be a group of* 3 *mutual friends or a group of* 3 *mutual strangers.*

Theorem 6.6 (Ramsey's theorem)**.** *For any natural number n, there is a natural number $R(n)$ such that if the edges of an $R(n)$-clique are colored red or blue, the resulting graph will contain either a red-n-clique or a blue-n-clique.*

Theorem 6.7 (The handshaking lemma)**.** *Each connected component of a graph G has an even number of nodes with odd degree.*

7 Induction

Theorem 7.1 (The principle of mathematical induction)**.** *Let P be some predicate. If $P(0)$ is true, and $\forall k \in \mathbb{N}.(P(k) \implies P(k+1))$, then $\forall n \in \mathbb{N}.(P(n))$.*

Definition 7.1 (Proof by induction)**.** To prove a statement of the form $\forall n \in \mathbb{N}.(P(n))$ prove:

- $P(0)$, which is called the base case.
- $\forall k \in \mathbb{N}.(P(k) \implies P(k+1))$, which is called the inductive step. The assumption of $P(k)$ for an arbitrary k is called the inductive hypothesis.

TODO: link to guide to induction.

Theorem 7.2 (Complete/Strong induction)**.** *Let P be some predicate. If $P(0)$ is true, and $\forall k \in \mathbb{N}.(\forall r \leq k.(P(r)) \implies P(k+1))$, then $\forall n \in \mathbb{N}.(P(n))$.*

8 Regular Languages

Definition 8.1 (Alphabet, string, language)**.** An *alphabet* Σ is a finite set of characters. A *string* over an alphabet Σ is a finite sequence of letters drawn from Σ. A *language* L over an alphabet Σ is a set of strings over Σ. The set of all strings composed of letters from Σ is denoted Σ^*.

Definition 8.2 (Empty string)**.** The *empty string* has no characters and is denoted ε.

Definition 8.3 (Language of an Automaton)**.** The language of an automaton D, is the set of strings it accepts, denoted by $\mathcal{L}(D)$.

Definition 8.4 (Regular language)**.** A language L is called *regular* if there is there is some DFA, D, such that $\mathcal{L}(D) = L$.

Definition 8.5 (Complement of a language)**.** The *complement* of a language L, denoted $\bar{L}$ is $\Sigma^* - L$.

Definition 8.6 (Language concatenation)**.** The *concatenation* of two languages L_1, L_2 denoted $L_1 L_2 = \{wx : w \in L_1 \wedge x \in L_2\}$.

Definition 8.7 (Language exponentiation)**.** Let $L^0 = \{\varepsilon\}$, then inductively define $L^{n+1} = LL^n$.

Definition 8.8 (Kleene Closure)**.** If L is a language over Σ, the *Kleene Closure* of L denoted L^* is $\{w \in \Sigma^* : \exists n \in \mathbb{N}.w \in L^n\}$. In other words $x \in L^*$ iff $\exists n \in \mathbb{N}.w \in L^n$.

Theorem 8.1 (Closure properties of regular languages)**.** *If L_1 and L_2 are regular languages, then*

- $\overline{L_1}$
- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 L_2$
- L_1^*

are all regular.

Theorem 8.2 (Subset construction)**.** *You can simulate any NFA with a DFA through a subset construction. Thus, a language L is regular iff there is a NFA N for which $\mathcal{L}(N) = L$.*

Theorem 8.3 (State elimination)**.** *The state elimination procedure can be used to convert a NFA into a regex.*

Theorem 8.4 (Equivalent definitions of regular languages)**.** *Let L be a language. The following are equivalent*

a.) L is regular.

b.) There is a DFA D for which $\mathcal{L}(D) = L$

c.) There is a NFA N for which $\mathcal{L}(N) = L$

d.) There is a regex R for which $\mathcal{L}(R) = L$.

9 Irregular Languages

Theorem 9.1 ($\{a^n b^n : n \in \mathbb{N}\}$ is not regular)**.** $\{a^n b^n : n \in \mathbb{N}\}$ *is not regular*

Definition 9.1 (Distinguishable)**.** x, y are *distinguishable* relative to a language L, denoted $x \not\equiv_L$ if there exists some $w \in \Sigma^*$ such that $xw \in L \iff yw \in L$. Namely exactly one of xw and yw is in L.

Theorem 9.2 (Myhill-Nerode Theorem)**.** *Let L be a language over Σ. If there is a set $S \subset \Sigma^*$ such that*

- *S is infinite*
- *For any $x, y \in L$, $x \neq y \implies x \not\equiv_L y$ (x is distinguishable from y).*

10 Context Free Languages

Definition 10.1 (Language of a CFG)**.** If G is a CFG, $\mathcal{L}(G) = \{w \in \Sigma^* : S \to^* w\}$.

Definition 10.2 (Context free language)**.** A language L is called *context free* if there is a CFG G such that $L = \mathcal{G}$

Theorem 10.1 (Every regular language is context free)**.** *Every regular language is context free.*

11 Computability Theory

Definition 11.1 (Church-Turing Thesis)**.** Every effective method of computation is either equivalent to or weaker than a Turing Machine.

Definition 11.2 (Encoding)**.** The *encoding* of an object x is some canonical 01-string representation. Denoted $\langle x \rangle$. You can also encode multiple objects denoted $\langle x, y, z, ... \rangle$.

Definition 11.3 (Accept, reject, loop, halt)**.** Let M be a Turing Machine.

- M *accepts* w if it enters an accept state after being run on w.
- M *rejects* w if it enters an reject state after being run on w.
- M *loops* infinitely on w if when run on w it never reaches an accept or a reject state.
- M *halts* on w if it either rejects or accepts w.

Note that if M does not accept w it doesn't necessarily reject w, it could also loop infinitely.

Definition 11.4 (Language of a Turing Machine)**.** The language of a TM, M, denoted $\mathcal{L}(M) = \{w \in \Sigma^* | M \text{ accepts } w\}$.

Definition 11.5 (Recognizable)**.** A language L is called *recognizable* if it is the language of some TM. The set of recognizable languages is called **RE**.

Definition 11.6 (Decider)**.** A TM, M is called a *decider* if it halts on all inputs.

Definition 11.7 (Decidable)**.** A language L is called *decidable* if there is a decider M such that $\mathcal{L}(M) = L$. Equivalently, L is *decidable* if there exists a TM, M such that

- $w \in L \implies M$ accepts w.
- $w \notin L \implies M$ rejects w.

The set of all decidable languages is denoted by **R**.

Theorem 11.1 (Universal Turing Machine)**.** *There is a TM, U_{TM}, that when run on an input of the form $\langle M, w \rangle$, simulates M on w, and does whatever M does on input w.*

Definition 11.8 (A_{TM})**.** $A_{TM} = \mathcal{L}(U_{TM})$, is the language of the universal turing machine. An equivalent definition is $A_{TM} = \{\langle M, w\rangle : M \text{ accepts } w\}$.

Theorem 11.2 (Self-Reference (Kleene's Second Recursion Theorem))**.** *It is possible to construct TMs that perform arbitrary computations on their own descriptions.*

Theorem 11.3 (Undecidability of A_{TM})**.** $A_{TM} \notin \mathbf{R}$*. A corollary to this is that* $\mathbf{R} \neq \mathbf{RE}$*.*

Theorem 11.4 (Undecidability of $HALT$)**.** $HALT = \{\langle M, w\rangle : M \text{ halts on } w\} \notin \mathbf{R}$

Definition 11.9 (Verifier)**.** A *verifier* for a language L is a TM, V, such that

- V always halts.
- For any $w \in \Sigma^*$ we have that

$$w \in L \iff \exists c \in \Sigma^*.(V \text{ accepts } \langle w, c\rangle)$$

A string c where V accepts $\langle w, c\rangle$ is called a certificate for w.

Theorem 11.5 (Verifiers and **RE**)**.** $L \in \mathbf{RE} \iff$ *there exists a verifier for* L*.*

Definition 11.10 (Diagonal Language)**.** The *diagonal language*, L_D, is the language containing descriptions of TMs that don't accept their own description.

$$L_D = \{\langle M\rangle : M \text{ is a TM and } \rangle M\langle \notin \mathcal{L}(M)\}$$

Theorem 11.6 (L_D is Unrecognizable)**.** $L_D \notin \mathbf{RE}$*.*

12 Complexity Theory

Definition 12.1 (Polynomial Time)**.** An algorithm runs in *polynomial time* if its runtime is some polynomial in the input length n. That is, time $O(n^k)$ for some constant k.

Definition 12.2 (P)**.** The complexity class **P** is the set of all problems that can be solved in polynomial time. That is

$$\mathbf{P} = \{L : \text{ There exists a polynomial time decider for } L\}$$

Definition 12.3 (Polynomial time verifier)**.** A *polynomial-time verifier* for a language L is a TM V such that V is a verifier for L and V runs in polynomial-time in the length of the input.

Definition 12.4 (NP)**. NP** is the class of problems that can be verified in polynomial time. Formally,

$$\mathbf{NP} = \{L : \text{ There is a polynomial time verifier for } L\}$$

Definition 12.5 (Polytime reducible)**.** If A and B are problems and we can solve A by transforming the inputs to A (using polynomial time) and using an algorithm for B on the transformed input, then we say that A is *polynomial-time reducible* to B, written $A \leq_p B$.

Theorem 12.1 (Reducibility)**.** *If $A \leq_p B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$. If $A \leq_p B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.*

Definition 12.6 (NP-hard)**.** A language L is called **NP**-hard if for every $A \in NP$, we have that $A \leq_p L$.

Definition 12.7 (NP-complete)**.** A language L is called **NP**-complete if L is **NP**-hard and $L \in \mathbf{NP}$.

Theorem 12.2 (A path to resolving P vs. NP)**.** *There is a* **NP***-complete language in* **P** *if and only if* $\mathbf{P} = \mathbf{NP}$*.*

Theorem 12.3 (SAT is NP-complete (Cook-Levin Theorem))**.**

$$SAT = \{\langle \varphi \rangle : \varphi \text{ is a satisfiable PL formula}\}$$

is **NP***-complete.*