Chapter 1. Basic Function Properties. (1) $F: S \to T$ and $G: T \to U$ are one-to-one, then $H: S \to U$ $H(s) = G(F(s))$ is one-to-one. (2) $F: S \to T$ is one-to-one, onto $G: T \to S$ exists where $G(F(s)) = s$ for every $s \in S$. (3) If $G: T \to S$ is onto then $F: S \to T$ exists where $G(F(s)) = s$. (4) Finite S, T: (a) $|S| \leq |T|$, (b) one-to-one $F: S \to T$, (c) onto $G: T \to S$. **Big-O Notation.** For $F, G: \mathbb{N} \to \mathbb{R}_+$, $F = O(G)$ if $a, N_0 \in \mathbb{N}$ s.t. $F(n) \leq a \cdot G(n)$. $F = \Theta(G)$ if $F = O(G)$ and $G = O(F)$. $F = \Omega(G)$ if $G = O(F)$. **Little-o.** $F = o(G)$ if for every $\epsilon > 0$ there is some N_0 s.t. $F(n) < \epsilon G(n)$ for every $n > N_0$. $F = \omega(G)$ if $G = o(F)$. **Topological Sorting.** G is directed graph. G is acyclic iff there exists a layering of G.

Chapter 2. Computation and Representation. Cantor's Theorem. The $\mathbb{R}$ are uncountable. There does not exist a one-to-one mapping $\mathbb{R} \to \{0,1\}^*$. **Cantor Power Set.** If A is any set, then $|A| < |P(A)|$. $\{0,1\}^* \to \mathbb{N}$ one-to-one. $\{0,1\}^\infty$ is $\{f : \mathbb{N} \to \{0,1\}\}$. $\{0,1\}^\infty \to \mathbb{R}$ one-to-one. **Prefix-Free Encoding.** For two strings y, y', we say that y is a prefix of y' if $|y| \leq |y'|$ and for every $i < |y|$, $y_i = y'_i$. Function E is prefix-free if $x, x' \in X$ do not exist s.t. $E(x)$ is prefix of $E(x')$. **FINITE COMPUTATION** **Chap 3. Defining Computation. Algorithm.** set of instructions for how to compute an output from input by following steps of "elementary steps." **AON Properties.** (1) commutivity, (2) associativity $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, (3) distributive $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$. **IF.** = OR(AND(a,b), AND(NOT(a), c)). **Circuits.** $f: \{0,1\}^n \to \{0,1\}^m$. C computes f if for every $x \in \{0,1\}^n$, $C(x) = f(x)$. **Circuits $\leftrightarrow$ Straight Line Programs.** $f: \{0,1\}^n \to \{0,1\}^m$, $s \geq m$, f is computable by Boolean circ of s gates iff f computable by AON-CIRC of s lines. **NAND.** NOT(a) = NAND(a,1). AND(a,b) = NAND(NAND(a,b), NAND(a,b)). OR(a,b) = NAND(NAND(a,1), NAND(b,1)). For every bool circ C w/ s gates, NAND-CIRC C' has at most $3s$ gates. **Equivalence.** NAND $\equiv$ {IF, ZERO, ONE}. AON-CIRC $\equiv$ IF-CIRC. Neither {AND, OR} or {XOR} is more powerful. {OR, NOT} $\equiv$ NAND. NAND > XOR. {MAJ$_3$, NOT, ONE} $\equiv$ NAND. NOR $\equiv$ NAND. **Chap 4. Syntactic Sugar.** [functions/macros, conditionals, bounded loops] $\to$ LOOKUP $\to$ compute every function. **NAND-CIRC-PROC.** defines non-recursive procedures. IF(cond, a, b) = NAND(NAND(b, NAND(cond, cond)), NAND(a, cond)). **LOOKUP Function.** For every k, $\text{LOOKUP}_k : \{0,1\}^{2^k + k} \to \{0,1\}$ is $\forall x \in \{0,1\}^{2^k}$, $i \in \{0,1\}^k$, $\text{LOOKUP}_k(x, i) = x_i$. Thm 4.10 For every $k > 0$, $\exists$ NAND-CIRC prog that computes LOOKUP (w/ at most $4 \cdot 2^k$ lines). **Universality of NAND.** Thm 4.12. $\exists c$ s.t. $\forall n, m > 0$, $f: \{0,1\}^n \to \{0,1\}^m$, NAND-CIRC prog w/ at most $c \cdot m \cdot 2^n$ lines computes f. Improved bound: $O(m \cdot 2^n/n)$. Proof w/ LOOKUP. $\therefore$ Every finite function computable w/ NAND-CIRC. **SIZE(s)** Def. $SIZE_{n,m}(s)$ = set of functions s.t. $f \in SIZE(s) = \bigcup_{n,m \leq 2^s} SIZE_{n,m}(s)$ = all functions w/ NAND-CIRC at most s lines. $|SIZE(s)|$ is smaller than the total # of functions mapping n bits to 1 bit. **Upper Bound** Thm 4.16 $\forall n, m > 0$, $f: \{0,1\}^n \to \{0,1\}^m$, $\exists$ bool circ $\leq c \cdot m \cdot 2^n/n$ lines ($c \leq 100$, arbitrary small $c > 0$) **Chap 5. Code as Data, Data as Code. Programs $\leftrightarrow$ Strings** for $f \in SIZE(s)$, $\exists P$ computing f whose string rep has $c \cdot s \log s$ lines ($c = 10$). **Counting Programs** $\forall s \in \mathbb{N}$, $|SIZE(s)| \leq 2^{c \cdot s \log s}$ ($c = 10$) functions computable by an s-line NAND-CIRC program. **Program $\leftrightarrow$ # lines lower Bound** For sufficiently large n, there exists a $f: \{0,1\}^n \to \{0,1\}$'s NAND-CIRC program requires at least $\delta \cdot 2^n/n$ lines ($0.1 < \delta < 1/2$). (if $f \in SIZE(\frac{\delta 2^n}{n})$) **TUPLE Representation** n inputs, m outputs: (n, m, L). L = list of tuples of NAND gates **Universality** Thm 5.10: **Universality of NAND-CIRC** $\forall s, n, m \in \mathbb{N}$ $\exists$ NAND-CIRC prog of at most $O(s \log s)$ lines that computes $\text{EVAL}_{s,n,m} : \{0,1\}^{1+n} \to \{0,1\}^m$. $\text{EVAL}_{s,n,m}(P, x) = \begin{cases} P(x) & P \in \{0,1\}^{S(s)} \\ 0^m & \text{otherwise} \end{cases}$, where $S(s) = 3s\lceil \log(3s) \rceil$ ← length of triple notation **Physical Extended Church-Turing Hypothesis** If a function $F: \{0,1\}^n \to \{0,1\}^m$ can be physically computed w/ s resources $\leftrightarrow$ boolean circuit w s gate **UNIFORM COMPUTATION**

Chap 6. Loops and Infinity. Turing Machines. Computable Functions $F: \{0,1\}^* \to \{0,1\}^*$, TM M. M computes F if $\forall x \in \{0,1\}^*$, $M(x) = F(x)$. F is computable if there $\forall$ a TM that computes it. **NAND-TM** $\equiv$ NAND CIRC + loops (arbitrary time) + Arrays (arbitrary sizes). **NOOP AND JUMP** (last line): $a=0, b=0$: halt. $a=1, b=0$: jump to first. $a=0, b=1$: jump to ith line and i--. $a=1, b=1$: jump to first line and i++. (i is unrestricted.) **Turing Equivalence.** TMs $\equiv$ NAND-TM programs: $\forall F: \{0,1\}^* \to \{0,1\}$, F is computable by a NAND-TM program iff there is a TM that computes it. **Uniformity.** A single algo can compute functions of all input lengths. **Chap 7. Equivalent Models of Computation. RAM Machines.** Allows for directly accessing memory locations. Can do: data movement, computation, and control flow. **NAND-RAM** $\equiv$ NAND TM + integer valued variables + indexed arr access + if/then + while/do. **TM $\leftrightarrow$ RAM Machines** Thm 7.1 $\forall F$, F is computable by a NAND-TM iff F is computable by a NAND-RAM program. Pf: Indexed access of bit arrays $\to$ 2D bit arrays $\to$ arrays of ints **Turing Equivalent models.** TM, NAND-TM, NAND-RAM, λcalc, Game of Life, Programming languages (Python, C, etc.). **Cellular Automata.** cell $\to$ new state w/ simple rule. **Game of Life.** dead $\to$ alive iff 3 neigh alive. alive $\to$ alive if 2 or 3 alive neigh. **Configuration of TMs.** Let M be a TM w/ tape alphabet Σ and state space $[k]$. A configuration of M is a string $\alpha \in \bar{\Sigma}^*$ where $\bar{\Sigma} = \Sigma \times (\{\cdot\} \cup [k])$ that satisfies that there is exactly 1 coordinate i for which $\alpha_i = (\sigma, s)$ for some $\sigma \in \Sigma$ and $s \in [k]$. For all other coordinates j, $\alpha_j = (\sigma', \cdot)$. A configuration $\alpha \in \bar{\Sigma}^*$ of M corresponds to the following state of its execution: (1) since α_j is a pair of alphabet symbol σ and either a state in $[k]$ or $\cdot$, α_{j0} is the first component σ of this pair. (2) M's head is in the unique position i for which α_i has the form (σ, s) for $s \in [k]$ and M's state = s. **Aka:** has (1) current head position, (2) full contents of large-scale memory (tape), (3) content of local registers (state of M). **NAND-TM config.** (1) current value of i (2) for every scalar var foo, the value of foo (3) for every arr Bar, Bar[j] for every $j \leq |t|$ where t = max(i). $\text{NEXT}_M : \bar{\Sigma}^* \to \bar{\Sigma}^*$ be the function that maps config of M to config at next step of execution. $\forall i \in \mathbb{N}$, $\text{NEXT}_M(\alpha)_i$ only depends on the coordinates $\alpha_{i-1}, \alpha_i, \alpha_{i+1}$. **1D cellular automata are Turing Complete** For every TM, there is a 1D cellular automaton r s.t. $E(\text{NEXT}_M(\alpha)) = \text{NEXT}(E(\alpha))$ $\forall$ config $\alpha \in \bar{\Sigma}^*$ of M, where $E(\alpha)$ is extension of α to r by defining $E(\alpha)_i = \alpha_i$ $\forall i \in \{0, \ldots, |\alpha| - 1\}$ and $E(\alpha)_j = \emptyset$ $\forall j$ s.t. $j < 0$ or $j \geq |\alpha|$.

Chap 8. Universality and Uncomputability

Universal TM U can evaluate all other machines, including machines more complex (w/ more states) than U. **Definition** $\exists$ TM U s.t. for every M representing a TM and $x \in \{0,1\}^*$, $U(M,x) = M(x)$. If M halts on x and outputs y, $U(M,x) = y$. If M doesn't halt ($= \perp$), $U(M,x) = \perp$. **TM → String** TM M has k states and size ℓ alphabet $\Sigma = \{\sigma_0 \cdots \sigma_{\ell-1}\}$. M = triple $(k, \ell, T = (\delta_M(1,0), \delta_M(0,\sigma_0) \ldots, \delta_M(k-1, \sigma_{\ell-1})))$. $\delta_M = (s', \sigma', d')$ where $s' \in [k]$, $\sigma' \in \Sigma$, $d \in \{0,1,2,3\}$ (L, R, S, H). $\therefore$ M is encoded by $2.38 \cdot \ell$ length list. **Uncomputable** **Uncomputable Exists** Diagonalization on inputs $\{0,1\}^*$ and $\{$programs $\}$. Define $F^*(x) = 1 - G(x)$ if $x = i$. **HALT** $\forall M \in \{0,1\}^*$, HALT$(M,x) = 1$ if TM M halts on input x and 0 o/w. HALT is uncomputable **Rice's Theorem.** **Semantic properties**: properties of the function a program computes as opposed to properties that depend on particular syntax used by a program. We cannot certify semantic properties of general purpose programs. **Semantic Functions** Definition 8.14, TMs M and M' are functionally equivalent if for every $x \in \{0,1\}^*$, $M(x) = M'(x)$ (or one halts iff the other does). $F: \{0,1\}^* \to \{0,1\}$ is semantic if for every pair of strings M, M' that represent = TMs, $F(M) = F(M')$. **Rice's Thm** Thm 8.15 Let $F: \{0,1\}^* \to \{0,1\}$. If F is semantic and nontrivial, it is uncomputable. **Chap 10. Is every thm provable?** **Mathematical Statements** Assert properties of any mathematical set, including sets, strings, functions, graphs, and programs. **Proof System** Definition 10.2 A proof for a statement $x: \{0,1\}^*$ is another piece of text $w \in \{0,1\}^*$ that certifies the truth of the statement asserted in x. (conditions: (1) **Efficiency**: $\forall x, w \in \{0,1\}^*$, $V(x,w)$ halts w/ output 0 or 1. Aka an algorithm exists for verification. (2) **Soundness** $\forall x \notin T$ and $w \in \{0,1\}^*$, $V(x,w) = 0$, aka there is a valid proof w for x then x is true. A true statement $x \in T$ is unprovable if for every $w \in \{0,1\}^*$, $V(x,w) = 0$. **Complete** V is complete if there $\nexists$ a x that is unprovable with respect to V.

Gödel's Incompleteness Thm. Thm 10.3 There does not exist a complete proof system for H (set of strings $x \in \{0,1\}^*$ representing TMs that halt on the zero input." **QIS** Def 10.6 is a well-formed statement w/ no unbound variables involving integers, variables, $<, >, +, -, \times, =, \neg, \wedge, \vee$, and quantifiers of the form $\exists_{x \in \mathbb{N}}$ and $\forall_{y \in \mathbb{N}}$ where x,y are var names. **Twin Prime conjecture.** ∞ number of p s.t. both p and $p+2$ are prime. **GIT for QIS.** Thm 10.8 Either V is not sound ($\exists^{\text{false}}_x, w \in \{0,1\}^*$ s.t. $V(x,w) = 1$) or not complete ($\exists^{\text{true}}_x$ s.t. for every $w \in \{0,1\}^*$, $V(x,w) = 0$). **MRDP Thm.** Thm 10.10 DIO: $\{0,1\}^* \to \{0,1\}$ takes string describing 100-var polynom w/ int coeff and outputs 1 iff $\exists z_0, \ldots z_{99} \in \mathbb{N}$ s.t. $P(z_0, \ldots, z_{99}) = 0$. Then DIO is uncomputable. **QMS.** Def 10.12 QIS with $\exists a \in \{0,1\}^*$, $\forall b \in \{0,1\}^*$, $|a| =$ length, $a_i =$ true if $i < |a|$ and $a[i] = 1$. **Uncomputable** Thm 10.13 Let QMS be the function that given str rep of QMS outputs 1 if true and 0 if false. Then QMS is uncomputable.

HALT ON ZERO uncomputable → QMS uncomputable → QIS uncomputable → Diophantine Eqns (MRDP) uncomputable

---

Proving Computability.

Computable: proof by Python

"Eat your cake and have it too": Computable by Python → computable by TM. Explain why the python function computes function.

Uncomputable: Reduction (non-semantic)

EX uncomputable functions: HALT, $F^*(x) = \begin{cases} 1 & x(x) = 0 \\ 0 & \text{o/w} \end{cases}$, HALTONZERO$(M) = \begin{cases} 1 & \text{if } M(0) \text{ halts} \\ 0 & \text{o/w} \end{cases}$

Assume by contradiction G is computable. P_G is a program that computes G. If $P_G \exists$, I can use P_G to compute uncomputable H.

Alg B. input: M, x goal: output 1 iff $M(x) \neq \perp$ def $P_H(M,x)$: resource: $P_G(N)$ $\forall N$ computing G

① def $N(w)$: transform $M \to N$ return
run M on x
return functionality of G

② return $P_G(N)$

We can use in all cases, $P_H(x) = \text{HALT}(x)$, which contradicts the fact HALT is uncomp.

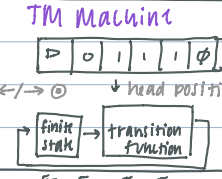
Uncomputable: Rice's Thm (semantic) Assume G is comp.

Let $\mathbf{0}$ be the prog that just returns 0. $G(M, \mathbf{0})$ is a semantic function. If M just rets 0, $G(M, \mathbf{0}) = 1$ if M doesn't halt, $F(M, \mathbf{0}) = 0$. By Rice's, $F(M, \mathbf{0})$ is uncomputable, which is a contradiction

Definitions. one-to-one = inj. onto = surj.

no map from $\{0,1\}^{80} \to \{0,1\}^*$ | **Prefix Free.** this diagram must be one to one.

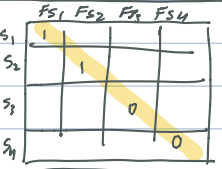
TM Machine



Eat your cake and have it too

- prove something can't be done w/ simple TM model
- prove something can be done w/ feature-rich RAM Machine

Cardinality of Int functions: Diagonalization



Each F_{Si} is defined on each string $\{0,1\}^*$ and the cells in the table represent the output of F_{Si}. We can construct a $G(S_i) =$ opposite of $F_{S_i}(S_i)$. G will differ from the output of F_{S_i} on the i^{th} string. $\therefore$ not onto because there is some G not in mapping.

XOR(a,b) = NAND(NAND$(a$, NAND$(a,b))$, NAND$(b$, NAND$(a,b))$)
= OR(AND$(a$, NOT$(b))$, AND(NOT(a), $b))$. 1 = OR$(a$, NOT$(a))$
0 = AND$(a$, NOT$(a))$

XOR $\subset$ NAND Proof. Induction on # lines of P. $k=1$, $F(x) = b + \sum_{i \in S \subseteq [n]} x_i \bmod 2$. of form $y_0 = \text{XOR}(x_\alpha, x_\beta)$. $k+1$ $F'(x) = b' + \sum_{i \in S' \subseteq [n]} x_i \bmod 2 = y_\alpha + y_\beta \bmod 2$. using

Analysis of Reductions

(1) $H(M,x) = 1$: $\forall N$ computes ①, returns $G(N)$ from ②.

(2) $H(M,x) = 0$: $w = \ldots$

$(a \bmod 2 + b \bmod 2) \bmod 2 = (a+b) \bmod 2 \to F'(x) = b_\alpha + b_\beta + b + \sum x_i \bmod 2$

If XOR could compute NAND, it has same form. NAND$(0,1)$ = NAND$(0,0)$, so $1 \notin S$. NAND$(1,0)$ = NAND$(0,0)$, so $0 \notin S = \emptyset$.

F is either constant 0 or 1 → contradiction ■.

MAJ cannot compute NOT. After n statements, only can compute $0, 1, X_i$ output

Logic Gates. Universal: AND/NOT, OR/NOT, NOR = NOT(x OR y), XOR/AND.

Not Universal: AND/OR, MAJ

Monotone: flipping input bit from 1→0 doesn't change 1→0.

Semantic F is not semantic $\neq$ F is computable. takes 1 TM

$EQ = \begin{cases} 1 & M \cong N \\ 0 & \text{o/w} \end{cases}$! $F(M) = EQ(M, N_0)$, $N_0(w) =$ XOR(bits in w)

$F = CX(M)$, which is semantic

LOOKUP. shows every finite func is comp. by NAND-circ → LOOKUP stores table of input and output

TM to NAND-TM.

head position → index
state space → scalar vars
tape/alphabet → arr vars
NEXT$_M$ → lines of NAND

SIZE. $SIZE_{n,1}(n) \subsetneq ALL_{n,1}$. Size Hierarchy Thm: $SIZE_{n,1} \subsetneq SIZE_{n,1}(n^2)$. $SIZE_{n,1}(n^2) \subsetneq SIZE_{n,1}(100 \cdot 2^n/n) = ALL_{n,1}$

NAND-TM Loops. blocks: fixed iteration w/ var and (active, final, initial flags/line)