Contents

1 Week 2

1.1 Pushdown Automata (PDA)

- L = { $a^n b^n$ | n $\geq$ 0 }
- 7 tuple, M = (Q, $\sum$, Γ, δ, s, F, $\perp$)
  - Q: Finite set of states
  - $\sum$: Finite input alphabet
  - Γ: finite stack alphabet
  - δ: transitions
  - $\perp \in \Gamma$
  - S $\in$ Q

- F $\subseteq Q$

- $\delta \subseteq (Qx(\sum$ U $\{\epsilon\}$ x Γ) x (Q x Γ*)((p, a, A), (g, $B_1B_2\ldots B_k$)
  - When in state p, reading in "a", and A is on top of the stack, the machine <u>may</u> move to state g, pop A off the stack and push $B_kB_{k\text{-}1}\ldots B_1$ (B_1 is now on the stop of the stack.)

1.1.1 Example

- Stage 1: Push A's
- Stage 2: Pop A's
- Stage 3: Are we accepting? (This mean B's and A's must be equal)

1.1.2 More examples

- a) L = { $a^n b^n c^n$ | n $\geq$ 0 } (Impossible)
- b) L = { ww^R | w $\in \sum^*$ } (Possible)
- c) L = { ww | w $\in \sum^*$ } (Impossible)
- d) L = { $a^n b^m c^m d^n$ | n, m $\geq$ 0 } (Possible)

1.2 Turing Machines (TM)

1.2.1 What can't PDAs do?

- Infinite memory, but it's not "perfect"
- What if we do have perfect/infinite memory
- Does this allow for more accepted languages
- How do we formally define it

1.2.2 Definition

- We'll say a TM is an 8-Tuple
- M = { Q, $\sum$, Γ, δ, s, qaccept, qreject, $\square$}
    - qaccept $\in$ Q
    - qreject $\in$ Q
    - qaccept $\neq$ qreject
    - $\square \in \Gamma$
    - $\sum \subseteq \Gamma$ - { $\square$}δ: Q$x\Gamma \rightarrow$ Q x Γ x {L, R}
    - $\delta(g_1, a) = (g_2, b, R)$

- If I am in g_1 and it is an a, write a b and move right

Input is initally written on the tape

If at any point the TM enters state qaccept, it stops (halts) and accepts the input

If at any point the TM enters state qreject, it stops (halts) and rejects the input

Can be used as language accepters, or as output machines

Does all input get accepted or rejected?

- No, it can loop

1.2.3 Example

- Construct a TM M sich that $L(M) = \{a^n b^n c^n \mid n \geq 0 \}$
- Start by scanning the input from L to R to ensure it's of the form a*b*c*
- Scan from R to L, overwrite exactly one c, b, the a with a special character
- Scan from L to R, overwrite exactly one a, b, then c repeat 2)
- If at any time, you encouter a letter out of order, reject the input

2 Week 3

2.1 Delta function layout

(1, a, R) - Move to state 1, if its an A move right

	a	b	c	-	$\square$
S	(1, a, R)	Reject	Reject	/	Accept
1	(1, a, R)	(2, b, R)	Reject	/	Reject
2	Reject	(2, b, R)	(3, c, R)	/	Reject
3	Reject	Reject	(3, c, R)	/	$(4, \square, L)$
4	Reject	Reject	(5, -, L)	(4, -, L)	Accept
5					
6					
7					
8					
9					
10					
11					

2.2 Output Turing Machine

2.2.1 Example

Let $\sum = \{a\}$. write a TM such that when M halts, the tape contains ww where w is the input. (Double the a's)

- a = aa
- aaa = aaaaaa

2.3 Language of a Turing Machine

- The collection or set of strings which M accepts L(M)
- A language L' is Turing-recognizable if $\exists$ a Turing machine M, L'=L(M)
  - w $\in$ L' $\Rightarrow$ if we run M on w, it accepts w
  - w $\notin$ L' $\Rightarrow$ if we run M on w, it rejects or infinite loops
- A language L¯ is Turing Decidable if $\exists$ a TM, M sich L¯ = L(M) and M halts on all input
  - w $\in$ L¯ $\Rightarrow$ running M on w accepts
  - w $\notin$ L¯ $\Rightarrow$ running M on w rejects

2.3.1 Question

- L is recognizable $\Leftarrow$ L is decidable
- L is decidable $\Rightarrow \forall$ turing machines M, such that L' = L(M) $\Rightarrow$ if M is run on w $\in$ L, M will halt? **NO**
  1. Let M be a TM which decided L'
  2. Make a new TM, M', which mimics M except when M rejects, M' will loop

2.4 Multi-tape Turing Machines

- Just like a classic TM, but it has k tapes
- δ: Q x $\Gamma \rightarrow$ Q X Γ x {L, R} is replaced with δ^{k}: Q X $\Gamma^{k} \rightarrow$ Q X Γ^{k} X $\{L, R\}^{k}$

2.4.1 Show a single tape TM can simulate a multi-tape TM

- Encode multiple tapes on a single tape

$\Gamma = \{a, b, \square\} \rightarrow \Gamma' = \{a, b, a', b', \square\}$

2.5 Single infinite tape TM

- Same as a classic TM but infinite only in one direction
- Just add some left padding
- This is like a 2 tape TM, the 2nd tape is the 1st one flipped

2.6 Enumerators

- 2 tapes and a control
  - Control has the output tape with the following conditions
    * Write Only
    * Tape alphabet = $\sum$
    * Only works to the right
  - Control has the work tape with the following confitions
    * read/write

        * Basically a TM
        * No accept or reject states
        * Enumerate state
    - Whenever the control reaches the enumerate state, it prints out whatever is on the output tape and clears it
    - L(E_ is the set of strings which W will eventually print out

3 Week 4

- I slept…

4 Week 4 Tutorial

4.1 Diagnolization to prove R does not map to N (Not countable)

- $d_{11}d_{12}d_{13}\ldots d_{1n}$ d_{21}……..d_{2n} ……….. d_{n1}……..d_{nn}
- Construct e such that e_{ii} = (d_{ii} + 2) % 10

4.2 Proof that $\mathbf{Eq_{BFA}} = <\mathbf{(DFA1, DFA2); } <\mathbf{(DFA1)} = <\mathbf{(DFA2)}>$

- 2·L(DFA1) = L(DFA2) $\Rightarrow$ (L(DF1) L(DFA2)) = ϵ AND (L(DFA1) L(DFA2)) = ϵ
- Checking if L(DFA) = ϵ is Decidable
    - Go through every state in the DFA
    - If you go to any state that is acccepting with a string that is not empty, reject
        * Otherwise accept.
- Full proof on pg 169 of tb
    - Introduction to theroty of computation 2^{nd} edition By M. Sipser

5 Week 5

5.1 Continuing..

- $A_{TM} = \{M\#_W \mid M \text{ accepts } w\}$
  - The membership/acceptance problem
  - Thm: A_{TM} is a recognizable language
  - Proof: Construct a new TM U
    - U = "on input $M\#_w$ where M is a TM and w is a string"
      1. Simulate M running on w
      2. If M accepts, U accepts
      3. If M rejects, U rejects
  - Universal Turing Machine (UTM)
  - Thm: M is undecideable
  - Proof: Assume that A_{TM} is decideable ⇒ there exost a TM D $D(M\#_w)$ = { accept if M accepts W, reject is M does not accept W } P = "on input M, where M is a TM:"
    1. Run D on input $M\#_M$
    2. Output the opposite of D
       - if D accepts, reject
       - if D rejects, accept
    3. P(M) = { accept if M does not accept M, reject if M does accept M }
    - What if we run P on itself? P(P) = { accept is P rejects, reject if P accepts } This is a paradox ⇒ Contradiction ∴ A_{TM} is undecideable
- Def Lbar = $\sum^*$ - L
  - Thm: A_{TM}Bar is unrecognizeable
  - Proof: Will follow from
    - Thm: A is decidable iff A and ABar are recognizable
    - Proof:
      1. Show A is decideable ⇒ A and ABar are recognizable
      2. Show A is decideable ⇒ A is recognizable
      3. ⇐ A is recognizable ⇒ ∃ TM m such that ∀w, M accept (and halt) w iff w ∈ A

        * Let M_1 and M_2 recognize A and ABar respectively $\Rightarrow$ A_W M_1 halts if w$\in$A, M_2 halt is w$\notin$A
        * M = "Run M_1 and M_2 in parallel"
            1. If M_1 accepts, accept
            2. If M_2 accepts, reject
        * M is a decider for A $\Rightarrow$ A is decideable
    - Thm (again): A_{TM}Bar is unrecognizeable
        * if A_{TM}Bar was recognizable $\Rightarrow$ A_{TM} is decidable, which it isn't

5.2 Halting Problem

5.2.1 Turings Method

- HP = { $M\#_W$ | M halts, on W }
- Thm: The halting problem is undecideable
- Proof:
    1. $\sum^*$ is countable
    2. The set of all TMs is countable
    3. Construct a TM P: P(w) = "on input w, construct M_w and we run D on $M_{w\#w}$
        (a) If D rejects, accept
        (b) if D accepts, loop
    4. Assume HP is decideable
        - D($m\#_W$) = { accept if M halts on w, reject if M loops on w }
        - P is not in my table, $\therefore$ contraduction.

5.2.2 Method 2

- Assume HP is decideable
    - Then $\exists$ a decider for HP

D_1($M\#_w$) = { "accept if M halts on w, reject if M loops on w" } D_2 = "on input $M\#_w$, M is a TM, w is a string"

1. Run D_1 on input $M\#_w$
2. If D_1 rejects, reject
3. If D_1 accepts, simulate M on w 3.1) If M accepts, accept 3.2) If M rejects, reject

$D_2(M\#_w)$ = { "if M loops on w reject, if M rejects w reject, if M accepts w accept" } D_2 decides A_{TM} A_{TM} = { $M\#_w$ | M accepts w } Contradiction, $\therefore$ HP is undecideable

6 Week 6

- Sick

7 Week 7

- Last Time
  - $A \leq_m B$
  - ES and REG are both undecideable
- Given a TM M
  1. Accepts any string
  2. Accepts the string OOOO
  3. Accepts every string
  4. Accepts a finite set
  5. Accepts a CFL
  6. Has 100 or more states
  7. Has more then 100 steps on input w
  8. 1 through 5 are undecideable
     - About the language of the TM (L(M))
  9. 6 through 7 are decideable
     - About the TM (M)

7.1 Anything about the language is undecideable

- Let S denote the set of all languages
- Let a property P be a subset of S
- A TM M has the property P if L(M)$\in$P
- Property
  - A TM accepts no strings $\Rightarrow$ $P_{\text{empty}} = \{ \ \}$
  - A property is non trivial if $\exists$ M_1 and M_2 such that $L(M_1) \in P$ and $L(M_2) \notin P$
  - $P_{\text{trivial}} = \{ L \mid L \text{ is recognizable} \}$
- Rice's Thm:
  - Let P be a non-trivial property of languages of Turing Machines
    * $L_p = \{ M \mid L(M) \in P \}$ is undecideable

7.2 Midterm

- Thursday March 1^{st}, 6:00-8:00pm
- IB120
- 5 Questions
- Topics
  - PDAs
  - Low Level TMs
    * Define a TM, states, transitions, alphabet, etc
  - Models of Equivalence
    * Ignore enumerators
  - Decideability and Recognizeability
  - A_{TM}/Halting Problems proofs
    * Don't memorize the proof in full
    * Know the idea, fill in the blanks/find the error
  - Reductions
    * Definitions

- * Proofs
    - · Show that language L is decideable/recognizeable using a reduction

- Define L is recognizable $\exists$ a TM M such that $\forall$w if w$\in$L then M accepts w and if w$\notin$L M loops or rejects w
- HP = { $M_{\#w}$ | M halts on w }
  - R($M_{\#w}$) = run M on w, if M accepts $\rightarrow$ accept. If M rejects $\rightarrow$ accept. If M loops $\rightarrow$ loop
  - $M_{\#w} \in$ HP iff M doesn't loop on w
- A is recognizable $\neg\rightarrow$ A' is unrecognizeable
- A is undecideable and A is recognizable $\rightarrow$ A' is unrecognizeable

7.2.1 Past test, quesiton 7

- L" = { M : L(M) = L_L }
- L_L = { $O^n l^m$ | n < m }
  - Oll $\in L_L$
  - O $\notin L_L$
  - $\epsilon \notin L_L$
  - lO $\notin L_L$
- is L" decideable, recognizable?
- is L"' decidable, recognizable?
- HP, A_{TM}, HP', A_{TM}'
- HP' $\leq$ mL
- Show L" is undecideable
  - HP $\leq_m$ L"
  - f($M_{\#w}$)$\in$L" $\Longleftrightarrow M_{\#W} \in$ HP
  - f($M_{\#W}$) = on input x, set x aside
    - * Run M on W

      * run a L_L recognizer on x
        - if it accepts → accept
        - if it rejcts → reject
  - $L(f(M_{\#W})) = \{ L_L$ if M halts on w, if M loops on W $\}$
  - $f(M_{\#W}) \in L" \iff L(f(M_{\#W})) = L_L \iff$ M halts on W $\iff M_{\#W} \in HP$
  - ∴ L"in undecideable

- if A $\leq_m$ B then A' $\leq_m$ B'
  - HP' $\leq_m$ L"'
  - ∴ L"' is unrecognizeable

8 Week 8

- What can (or can't) camputers do (effectivness)
- So something using (relatively) little resources
  - Time and space

8.1 Moving Forward

- Look at low level TM
- Compare Models
- Classify problems / Heirachy - Reductions

8.2 The Lecture

- Def Let M be a total deterministic TM, the time complexity of M is a function f:N → N, f(n) is the max number of steps M uses on any input of length n
  - M is a f(n) TM
  - M runs in f(n) time
  - Def f, g $N \to R^T$, $f(n) = O(g(n))$ if $\exists n_0$, c $\forall$ $n>n_0$: $f(n) < cg(n)$

- Def f, g: N→R* f(n) = o(g(n)) if

$$lim \frac{f(n)}{g(n)} = 0$$

g grows <u>faster</u> then f eventually

- Facts
  - f(n) = O(f(n))
  - f(n) $\neq$ o(f(n))
  - f(n) = O(g(n)) ¬→ f(n) = o(g(n))
  - f(n) = o(g(n)) → f(n) = O(g(n))

- Consider the language L = {$0^n 1^n$ | n≥0 } M_1 = on input w
  1. Scan from L to R
  2. Cross off first O encountered 2,1) Then cross off the first 1 encountered
  3. When is reached, go back to the begining
  4. Repeat 2 4,1) If anything is found out of place, reject 4,2) If no 0 or 1 before , accept

- M is a f(n) = o(nlog(n)), then L(M), is regular
  - Def: t: N→R^t, the time class TIME(t(n)) is the set of all languages that are decidable by a O(t(n)) TM.
    - is L ∈ TIME(N^3)?
    - is L ∈ TIME(N)?
    - is L ∈ TIME(nlog(n))
  - Thm Let t(n) be a function where t(n)≥n. Then every t(n) time multitape TM has an equivalent O(t^2(n)) single tape TM
    - How long can the single tape be at any moment in time? Assume t(n)≥n
      - Each multitape can be at most t(n) long
      - The single table can be at ≤ k*t(n) + k - 1 (Account for the boundry characters) = O(t(n))

- In the worst case for a single "step" take

$$O(t(n)) + kshifts$$

$$O(t(n)) + k * O(t(n))$$

$$O(t(n))$$

- Number of steps to simulate = t(n)
- ∴ Total simulation time t(n)*O(t(n)) = O(t^2(n))
- Let N be a total NTM, its running time f(n) is the running time of itslongest branch
- Thm Let t(n)≥n, then every t(n) NTM has an equivalent $2^{O(t(n))}$ time deterministic TM
- Def P is the class of languages that are decideable in polynomial time on a deterministic single-tape TM.

$$P = U_{k=0}^{inf} TIME(n^k)$$

9 Week 9

- All deterministic computational models are polynomial equivalent
  - If you can code a program which runs in polynomial time, a single table TM exists, which does the same thing

9.1 Example

PATH = { <G, s, t> | There's a path from s to t in G }

- Theorem PATH ∈ P
  - Proof A poly time alporithm M operates as follows M(G, s, t) =
    - "Mark" s
    - While at least one more node has been marked since last iteration
      - For each edge (a, b) if a is marked and b is not, mark b
      - If t is marked → accept, else reject

9.2 Example

HPATH = { <G, s, t> | there's a Hamiltonian path from s to t }

- Def A Hamiltonian path goes through every other node exactly once
- Def A verifier for a language L is an algorithm V where:
  - L = { w | V accepts <w, c> for some c }
  - c is called the certificate
  - A poly time verified runs in polynomial time in the length of w
  - A language L is polynomial time verifiable if it has a polynomial time verifier (This TM always halts)
- Def One NP is the class of languages which have polynomial time verifiers
- Def Two A language is in NP iff it is decided by some polynomial time NTM

9.3 Question

If a language is in P, it is also in NP?

- Def One
  - V(<w, c>) = run D_{poly} on w
  - D_{Poly} = decider for L
- Def Two

– { TM } $\subseteq \{NTM\}$

9.4 Question

If a language is in NP, it is also in P?

- Well this is the P = NP problem. We know sometimes, not if it is always.

9.5 Complexity Reductions

- Def Language A is a polynomial time reduceable to language B, A $\leq_p$ B, if $\exists$ a polynomial time computable function f, where $\forall$w, w $\in$ A $\iff$ f(w) $\in$ B
  - Typical use Given problem A, assume we have a black box which solces problem B. In polynomial time, make your instance of A look like an instrance of problem B. Run your B black box on that instance, return an answer appropriatly
- Example LONG = { <G, s, t, d> | $\exists$ a path from s to t in G, with at least distance of d }
  - No cyles allowed, can't visit a node more then once
  - Prove that HPATH $\leq_p$ LONG
    1. Assume I have a black box which solved LONG
    2. To solve HPATH(G, s, t)
       (a) Set all edge weights to 1
       (b) Call LongSol(G, s, t, |v| - 1)
         * If yes $\rightarrow$ yes, if no $\rightarrow$ no
  - Thm If A $\leq_P$ B, and A $\in$ P, then A $\in$ P
  - Caollary If A $\leq_p$ B and A $\notin$ P, then B $\notin$ P

10 Week 10

10.1 From last time

- A $\leq_p$ B
  - If we can solve B, then we can solve A
  - Then B is at least as hard to solve as A is

10.2 Toolbox of problems

- Vertex Cover Given a graph G=(V, E), a Vertex Cover, C, is a set of verticies such that $\forall(v_1, v_2) \in$ E, $v_1 \in$ C, $v_2 \in$ C
  VertexCover = {(G, K) | G has a vertex cover of size k or less }

- Independent-Set Given a graph G=(V, E), an independent set is a set of verticies I, such that $\forall(v_1, v_2) \in E, \neg(v_1 \in I$ and $v_2 \in I)$
  Independent-Set = {(G, K) | G has an independent set of size K }
  - Prove: Independent Set $\leq_P$ Vertex Cover Let C be a vector cover of G=(V, E). Consider any two nodes v_1 and v_2 in V-C.
    What if v_1, $v_2 \in$ V-C and $(v_1, v_2) \in E \Rightarrow$ C is not a VC
    * $\therefore$ v_1, $v_2 \in$ V-C we know $(v_1, v_2) \notin E$
    * $\therefore$ V-C is an independent set in G
    * $\therefore$ If there is a vertex cover of size |V| - k in G, then there's an independent set of size k in G

    IS(G, K) = call VertexCoverSol(G, |V|-k)
    * If accept $\rightarrow$ accept (yes)
    * If reject $\rightarrow$ reject (no)
- Clique Given a graph G=(V, E), a cliquw of the graph is a set S such that S$\subseteq Vand\forall$ v_1, $v_2 \in S$, $(v_1, v_2) \in E$ or $v_1 = v_2$.