CS 172: Complexity Theory and Computability

Lecture Notes of Alistair Sinclair

Surya Vengadesan

March 8, 2021

Contents

1 Format of Notes

I apologize for the delayed update. I'll add both lecture notes and textbook notes. I will attend live lecture, and attempt to update the material by the end of that day. Afterwards, I will go back at add edits to handle typos and poor formatting. I'm currently experimenting, and over time, the formatting so should become better, more readable, and include more graphics. I will try to transcript the key topics and ideas, then afterwards, provide you with a summary of how all the information ties in together.

2 Logistics

Class site: `https://people.eecs.berkeley.edu/~sinclair/cs172/s21.html` Professor Site: `https://people.eecs.berkeley.edu/~sinclair/#grads` HW Problems
HW 1: Constructing DFA, Converting DFA to NFA, Proof using Induction why a specific DFA accepted a specific language, Regular language proofs
HW 2: Regular Expressions, Converting Regular expressions to DFA

Discussion Problems
Disc 1: Constructing finite automatons
Disc 2: Converting NFAs into regular expressions

3 Tuesday, January 19, 2021: Lecture 1

3.1 1st Pass: Key Points

Alistar Sinclair

sinclair@cs

OH M 9-10, Tu 1-2

testing parskip

Sara Fridovich-Keil 3rd Year PhD Student sfk@eecs OH M 5-6 Th 3-4

Exam Thursday 25th Feb
Thursday 2xth April
Thurday Final's Week 8-11am May

Grading
MT1 - 20
MT2 - 20
Final - 40
HW - 20

Read material before lecture
Read material again after lecture
Go to discussion each week

Deadlines
HW due Thursday Night

Second Half of Lecture after break was not recorded, because the professor lost Electricity

4 Thursday January 21, 2021: Lecture 2

4.1 1st Listen: Key Topics

Takeaway of big ideas, 2x speed
1. Read Course Policies
2. Turn on camera
3. Request to come to live lecture if you are in this timezone
4. Discussion sections on 2-3pm on Thursday and Monday 4-5pm
5. Above are just logistics
6. Goals:
7. Overview of class and Finite Automata
8. Look at different models of computations
9. Algorithms focuses on what can be done
10. This class is about what cannot be done
11. Appreciates the use of algorithms once you understand this
12. Engineering out of Physics
13. Computing is out Theory of Computation
14. Theory of Computation started with Turing in 1936
15. Three Parts
16. 1. Restricted models
17. 2. Finite automata - finite amount of memory, with restricted input
18. 3. Used to model the human brain
19. 4. Used as lexical analyzers but can't parse parentheses
20. 5. Streaming algorithms
21. 6. Pushdown automata, uses FA + stack (last in first out)
22. 7. Compilers covers pushdown automata
23. 8. 2nd Model: General Models
24. 9. Turing Machine
25. 10. Machine is unbounded memory
26. 11. Church-Turing Thesis
27. 12. Non-computable problems
28. 13. Godel's Incompleteness Theorem
29. 14. 3rd Model: Resource-Bounded Computation
30. 15. P: Polynomial-time solvable problems
31. 16. Extended C-T Thesis
32. 17. Quantum modeling?
33. 18. P = NP, Search algorithms
34. 19. PSPACE
35. 20. LOGSPACE
36. 21. Polynomial Hierarchy
37. 22. Provable intractable problems
38. 23. Zero Knowledge Proofs

39. WARNING: Proofs
40. Less about algorithm design, and more about understanding
41. Copying styles of proof is good
42. If proof is correct, you should be able to keep them short
43. Proofs in lecture, aren't model proofs, because professor knows the proved before hand
44. Question on Proofs: What is readable? What is the creative process? Someone says they don't know how to read a proof, you remember that and try it something again. Define my terms
45. Finite Automata
46. McCullouch & Pitts
47. finite set of states, with finite input, processed as stream
48. Example: Vending machine
49. Accepts quarters and dollar bills
50. Won't accept more than 1.25, and each soda is 1.25
51. Diagram of state diagram by Eric Gribkoff
52. Alphabet: 25, 1, select
53. States: 0, 25, 50, ..., 200
54. Transition Function: delta: Q x Sigma maps to Q
55. Start state is also accepting state
56. Example 2:
57. Moore Mealy Machines
58. Example 2: Farmer, Wolf, Goat, Cabbage
59. Man cannot leave pairs together
60. There is a state diagram
61. There is a dead state
62. Shortest paths
63. Backtracking is also possible
64. Def: Finite Automata is a 5-tuple
65. M = Q, Sigma, q zero, F, delta
66. states, alphabet, starts state, accepting or final states, transition function
67. Language accepted by the machine is L of M
68. all finite strings over alphabet, such that machine ends in an accept state
69. Computation: is a sequence of states
70. Delta star is the function of state of Machine after reading entire string
71. Example
72. Input state needs at least of one 1, where it on the left hand side if even number of zeros
73. Rigorize meaning of machine
74. Different accept states would allow for different languages
75. Example
76. 0-1 String that contains 010
77. Example
78. Arithmetic example

79. 0-1 String that are binary encoding of multiples of 3
80. Study the residue classes
81. Example
82. English spelling rule of i before e
83. Go to Alistar's OHs to ask more questions

4.2 2nd Read: Big Ideas

Summary of Lecture:
We covered the key ideas ideas and models of computation that will be covered in the course. This includes Restricted Models, Generalized Models, and Resource Bounded Models. Each subgroup goes deeper, where for example finite automata is a subset of Restricted Model due to it's finite memory. General models include Turing machines, and then Resource Bounded models cover the general area of PSPACE, LOGSPACE, and the extended Church-Turing thesis. Again, the lecture was filled with buzzwords of what the breakdown of the class for the foreseeable future. We then walked through the definition of finite automata, then went through examples of it applied in the vending machine, the river riddle, and then on generic bit strings with different properties. The examples of bit strings are found in the book, and I will incorporate them into the notes, once I figure out how to improve my setup for drawing finite automatas in latex.

4.3 Textbook Notes: S 1.1

1.1 Finite Automata
Computational models - exist with differing levels of accuracy (e.g. parallel to Scientific Models) Finite State Machine - (i.e. Finite Automaton) simplest CM, models C (e.g. electromechanical devices) with small M (i.e. memory) Markov Chains - FSM's probabilistic counter parts

Example: Automatic Door Controller
States: Open, Closed Actions (i.e. Input Conditions): Front, Rear, Both, Neither Representations: State Diagram and State Transition Table

Formalizations: states, start state, accept state, transitions, accept, reject, input string

Automaton starts at A, moves to

Formal Defintion of Finite Automata
5 Tuple Transition function language of machine A M recognizes A (i.e. M accepts A) empty language $\emptyset$ empty string ϵ

Formal Defintion of Computation
Machine M accepts string w if sequence of state $r_1, ..., r_n$ in Q satisfy three conditions:

1. $r_0 = q_0$ 2. $\delta(r_i, w_i + 1) = r_{i+1}, for i = 0, ..., n-1$ 3. $r_n \in F$

Definition 1.6: Language is regular language if finite automaton recognizes it

The Regular Operations
Objects: Languages Operations: union, concatenation, star

4.4 2nd Read: Big Ideas

Summary of Notes: Here we define finite automatons using it's formal tuple definition, then define what computation means using another formal definition. We then go onto defining languages based on finite automatons. Specifically, a regular language is defined to a language recognized by a DFA (deterministic finite automaton). Finally, we define regular operations, which are operations over regular languages, which included the union, concatenation, and star operator.

5 Tuesday, January 26th, 2021: Lecture 3

5.1 1st Listen: Key Concepts

Logistics
HW 1 is the Friday of this week
Midterms will still be in-class
No heavy duty exam proctoring, with just Zoom camera
Maybe open book, but not open internet

Topics:
Review of previous class, and cover sipser 1.2
Recovered tuple definition of finite automata
Recovered notation of sigma star
Recovered definition of computation
Recovered definition of delta star
Recovered example of 0-1 string with 010
Deterministic finite automata has alphabet number of systems coming out of it

New Material:
Non-Deterministic Finite Automata
FA with multiple or zero transitions for each character
Allow epsilon transitions, transition without even readings string
New definition of accepts
ND - think about what states you COULD be in

Example
We build a NDA for string divisible by 2 or 3
Epsilon transition to the different states
Fact: For every n, we can design a language that is accepted by NFA with n states, but the smallest DFA has 2 to the n states

Two views of nondeterminism
1. Subsets: NFA is in the subset of all the states
2. Tree: Tree of computations
Accepting, if at least one leaf is in accepting state

Epsilon used both for empty string and empty transitions

Will prove that NFA can be written as DFA

Theorem: The classes of languages by DFAs and NFAs are the same
Note: We can these regular languages
Proof:
Forward: L accepted by DFA, implies L accepted by NFA is trivial, since all DFAs are subsets of NFAs by construction
Backward: L accepted by NFA, implies L accepted by a DFA
Define: New DFA M' with Q' = P(Q) which is subset construction, q zero prime is q zero, F' is set of all subsets containing element of F, and delta prime is union of epsilon-closure of elements of subset with respect to an alphabet letter
$M' = Q', \Sigma', \delta', q_0', F'$
$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$ $E(R) = q$—q can be reached from R by traveling along 0 or more ϵ arrows
$E(R)$ applied to transition function and starting state. Therefore, the ϵ arrows from the NNFA are accounted in the DFA, completing the construction.

Example
We walk through an example of converting an NFA into a DFA
This is found within the book
Professor recommends walking through this specific example from scratch yourself

Question: How do we implement a FA?
DFA: Some for loops
NFA: Convert to DFA, but this may blow up
Instead, you use dynamic programming to maintain the set of states the DFA can currently be in
Maintain Q sub t to be the set of states after reading the first t input symbols
Update by including all states reachable from a state in Q sub t on reading symbol and epsilon-transition
Next Lecture: Regular expressions, read Sipser 1.3

5.2 2nd Pass: Big Ideas

During lecture, we introduced the topic of epsilon transitions and NFA. Before that, we did a quick review of the notes from the previous chapter. We went over an example of showing how an NFA with epsilon transitions can be used to define a NFA that reads mod 2 and mod 3 bit strings. We then covered two ways of thinking about an NFA, as both a powerset of subsets, but also as a tree. We then prove than an NFA can be written as a DFA forwards and backwards. We also work through an example of the powerset of subsets construction with an example. We ended the lecture discussing how to model a FA, both a DFA and NFA with dynamic programming.

Biggest takeaway: Nondeterminism, so read 1.2 from the textbook.

5.3 Textbook Notes: S 1.2

Nondeterminism viewed as parallel computation, with ”processes” or ”threads”
NFA corresponds to a process ”forking” into several children
NFA corresponds to tree where root is start of computation and each branching point corresonds to branching points with multiple choices

6 Thursday January 28th, 2021: Lecture 4

Logistics
HW 1 due tomorrow of this week, and HW due the friday of the next week
Discussion sections
No conflicts for final, and drop deadline for class

Review: Nondeterminism

New Topics:
closure operations in regular languages
regular languages

Suppose languages A, B are regular
What can we say about their operations? Will they be regular?
Union
Intersection
Complement
Concatenation
Star
Reverse
Note: there is a set notation for each one
Note: all will output regular languages

Claim: If A,B regular =¿ A union B regular
Proof: Given separate DFAs, construct new FA for A union B
Used set notation
Cartesian product of state space
tuple of start states
delta and accepting states defined

Proof: Using NFA
I don't understand this
Find it in the book

Complement
Claim: A is regular, implies A complement is regular
Proof: Flip accepting and non-accepting states
Notation: F' -¿ Q - F
NFA Proof: all leaves are accept or reject

Flip operation: if there was at least one accept and one reject, the same will be true for the flip
Concatenation
Claim: If A and B are regular then A concat B is regular
Proof: Accept state of the first machine connected to start state of next step with epsilon transitions

Reverse Proof
Left as an exercise and covered in discussion

Star Operator
Claim: A is regular, implies A star is regular
Proof: The epsilon transition is set of all the accepting states to start state. Set new start state to old start state with epsilon transition

Question: When do you use DFA or NFA in proofs?
Ans: To prove regular, then use DFA, but if you want to show current language transformed into another regular language, then NFA used

Question: Infinite union a regular language?
Answer: No? Counter Example, L sub k is the set of w containing k 0s and k 1s.
L sub k is regular for every k
Union of all L sub k is not regular because w has some number of 0s and 1s

Break in Middle of Lecture

Regular Expression over Alphabet
1. Atomic expressions
Empty set, empty string, set a of each element of alphabet
2. Given regular expressions R sub 1 and R sub 2, we build new ones
R sub 1 union plus R sub 2
R sub 1 concatenated R sub 2, equivalent to R sub 1 R sub 2
R star

Examples
Sigma star, shorthand for finite strings of all elements of alphabet
We need to get comfortable writing in regular expression format

Facts:
epsilon concat language R = language R concat epsilon

Theorem: A language L is regular (i.e. recognized by a FA) iff L = L(r) for some regular expression.
Proof: Use induction the structure of R.
Base Case: Atomic expressions
R = empty, epsilon, and a
Induction: Closed under union, concat, and star
Forward: Given FA, construct a regular expression that denotes some language
Generalize: NFAs to GNFAS, same except that transitions can be labeled by any regular expression
Normal FA are special case of GFDA
Assume:
start state has transition to every other state and from any other state
unique accepting state that has transitions from every other state and to no other state
every other pair of states has a transition between them in both transitions

Repeated remove states, one at a time, until we are down to start to end state
L(R) = L(M) maps to output R

Removing state q
Goal: Remove state q sub i to q sub j
Try different algorithms that generates regular expressions in place of removed states

He does not state what the next lecture will be on

6.1 2nd Pass: Big Ideas

While we focused on non-determinism on the previous class, we are now studying the relationships of regular expressions with DFAs and NFAs. We introduce regular languages and six operations on them: union, intersection, concatenation, star, reverse, and complement. We go over the proofs of all of them except reverse. The proofs are formulated solely by the use of clever manipulations of the input FAs of the languages into another FA using epsilon transitions and new states. The first proof on union requires that we define a new transition function over a new FA that is the pair of all possible tuples of the initial state spaces. We also go over the complements proof right afterwards. Then start on regular expressions, define atomic expressions, then how to build new languages with regular expressions. We also cover generalized deterministic finite automata (GDFA), then proceed to provide a proof on them I believe. My textbook notes will be more thorough. The details of the proofs in lecture are hard to grasp, and going back to the textbook where the professors get his examples will cement the material.

6.2 Textbook Notes: S 1.3

Equivalence with Finite Automata:
Regular expression and finite automata are equivalent in descriptive power
Regular expression that describes a language, can be converted into DFA

Theorem 1.54
Lemma 1.55: Forward Direction
If a language is described by regular expression then it is regular
Idea: Convert regex into NFA first
1. Convert Regex R = a into NFA
2. Convert Regex $R = \epsilon$ into NFA
3. Convert Regex $R = \emptyset$ into NFA
4, 5, 6. Show union, concat, and star of R's is another regex R
Use appropriate closure constructions proofs for 4, 5, and 6

Conceptual Questions:
How do the proofs in lecture work?

7 Tuesday February 2nd, 2021: Lecture 5

7.1 Live Lecture

LOGISTICS Homework due Friday
Waitlist all cleared
Move from Thursday to Monday section to load-balance

OVERVIEW
Finish FA to Reg Exp Proof
Non-Regular Langauges Proof

Theorem
Fix alphabet Σ. Language $L \subseteq \Sigma*$ is recognized by a DFA/NFA $\iff$ it is denote by a regular expression.

Proof: <= Proven. Remember regular languages are closed under $\cup$, '$*$. Given base cases of fundamental elements, induction proves the rest.
=> Given any regular language (by a FA M), we need to construct a regular expression that denotes L.

Work with GNFA model - like NFAs but with regular expressions on transitions
Assume start state has transitions to ever other state and from no state. Accept state is unique and has transition from every state and to no state. Every other pair of states has a transition between them.

ALGORITHM
Repeatedly remove the state, but don't remove any associated paths
Start state q_0 and end state q_e.
Remove one state at a time until you just have start and end state.
There will be one huge regular expression left on the remainin path.
End up with L(R) = L(M)

STEP: (REMOVING ONE STATE)
Note: never remove start or ending state, because we will remove the path that connects the two.
We want to remove state q.
$Q\ q1, q$ -¿ q -¿ $Q\ q, q_s$
Split the graph into three sets as describe above.
Find new transition label from q_i, q_j.

New $q_i- > q_j$ becomes $R_4 \cup R_1 R_2 * R_3$.
Question: If any of R 1,2,3,4 was the empty set, then what would happen?
Answer: If R_1 is empty, then the whole set is empty. But the self-loop of R_2 is important.
Question: What if q_i and q_j have no intermediate states?
Answer: We can still perform a transformation.

EXAMPLE:
DFA that reads binary numbers that are divisble by 3
Step 1: Redraw Machine with new start state and accepting state q_s, q_f, with ϵ-transitioin from start and to accepting state.
Step 2: Remove state 1 first.
Redraw state 1 in the form as describe above.
Determin new regular expression from state 0 to state 2.
Forward transition = 10, Backward transition 01, Selfloops $00 \cup 1$ and $11 \cup 0$.
Step 3: Remove state 2 next.
Note: each state removed makes regular expressions more complicated.
Redraw state 2 in the form above.
New path $10(00 \cup 1) * 01 \cup (11 \cup 0)$.
Step 4: Remove state 0 next.
Path from q_s to q_f is $R = (10(00 \cup 1) * 01 \cup 11 \cup 0)*$.
We don't know if this regular expression is in simplest form.
$L(R*)$ = concatenations of zero or more strings from L(R)

Question: Are GNFA's useful beyong this context?
Answer: It is used currentlly as a proof tool, and can be extended to other proofs.

BREAK: 11:58-12:02

Logistics: HW 1 has been graded.
HW1 Solutions have been posted, where there are alternative ways of doing things.
If you have questions about psets, go to office hours.

TOPICS: Non-regular languages
1. Can we show some languages are not regular? (i.e. Taks which FA can't do)
A: Find the total number of zeros, can't because we don't have infinite memory.
2. Can we find a minimal DFA for a regular language?
A: We first need to find a structure of the minimum DFA

Reading: Note 1 (replaces Sipser 1.4)

Pumping Lemma is not intuitive, but feel free to read it outside lecture

Distinguishable Strings

Def: Let L be a language over Σ. Two string $x, y \in \Sigma*$ are distinguishable w.r.t. L if $\exists z \in \Sigma*$ s.t. $xz \in L and yz \notin L$ (or vice versa).

EXAMPLE: $L =$

8 Thursday February 4th, 2021: Lecture 6

8.1 Live Lecture: First Pass

LOGISTICS:
Section starts at 2pm Thursday
HW2 due Friday 5pm
HW1 graded and on Gradescope
Regrades open Friday noon, close Monday noon

TODAY:
Minimizing DFAs

REVIEW:
Definition: Two string $x, y \in \Sigma*$ are distinguishable w.r.t. L is $\exists z \in \Sigma*$ s.t. $xz \in L$ and $yz \notin L$ (or vice versa)
Set $S \subseteq \Sigma*$ is distinguishable w.r.t L if every pair $x, y \in S$ is distinguishable

Fact: Any DFA of M for L must end up in different states on distinguishable inputs x, y
Corollary: IF $\exists$ set of k distinguishable strings for L, then any DFA for L must have at least k states
if $\exists$ infinite set of distinguishable strings w.r.t L, then L is not regular

e.g. non-regular languages: 0-1 strings with equal no. of 0's and 1's balanced string of parentheses
Doesn't work because machine needs to record arbitrarily long string

Observation: Define $x\ _Ly$ iff x, y are not distinguishable w.r.t. L, then $\ _L$ is an equivalence relation

Proof: (i) $x\ _Lx$
(ii) $x\ _Ly$ =¿ $y\ _Lx$
(iii) $w\ _Lx$ and $x\ _Ly$ =¿ $w\ _Ly$
Suppose: $w\ _Ly$. Then $\exists z$ s.t. $wz \in L$ $yz \notin L$, then must have either $xz \in L$ or $xz \notin L$, so either w_Lx or x_Ly

So $\sim_L$ partitions $\Sigma*$ into equivalence classes of indistinguishable strings
Any DFA for L must have at least one state per equiv class
Any DFA for L must have at least one state per equiv class
Another equiv relation for a DFA M: $x \sim_M y$ iff $\delta * (q_0, x) = \delta * (q_0, y)$

Fact: $\sim_M$ must be a refinement of x_L

Theorem [Myhill - Nerode]: Let L be any language over Σ. If $\sim_L$ has infinitely many classes then L is not regular. Otherwise, L is recognized by a unique minimal DFA whose no. of states is equal to the no. of equiv. classes of $\sim_L$

Proof: $\sim_L$ infinitely many classes =¿ L is not regular
So asuume $\sim_L$ has k equivalence classes.
Define a DFA $M = (Q, \Sigma, \delta, q_0, F)$ with one state per equivalence class. Use notation $[x]$ to denote the equivalence class containing string x.
Start State: $[\epsilon]$
Accepting State: $[x]$ for any $x \in L$
Transitions: $\delta([x], a) = [xa]\ \forall a \in \Sigma, \forall x$
With this def: an input $x = x_1 \cdots x_n$
$[\epsilon]- > [x_1]- > [x_1, x_2]- > [x_1, x_2, x_3]- > \cdots - > [x]$, accepting iff $x \in L$

Need to check above is well-defined, i.e. need to check: $x \sim_L x'$ =¿ $[xa] = [x'a] = xa \sim_L x'a$

Proof: Suppose $xa_L x'a$ - then $\exists z$ that distinguishes them i.e. xaz, $x'az$ are note both in L or out of L. But then az distinguishes x, x' so $x_L x'$. $\square$

Example: $L =$ 0-1 strings containing 01
Equivalence classes $[\epsilon]$, $[0]$, $[01]$
$[01]$ = x that contain 01
$[0]$ = x that don't contain 01 and end with 0 e.g. $1 * 00*$
$[\epsilon]$ = x .... and don't end with 0 e.g. $1*$

Goal: Given a DFA M, find the minimal automaton that recognizes $L(M)$
Definition: States $p, q \in Q$ are equivalent w.r.t M (written $p \equiv_M q$) iff for every $x \in \Sigma*$, the states reached by M starting from p and q are either both accepting or both non-accepting

If $p \equiv_M q$, then should be able to merge them into one state

Definition: For $k \geq 0$, define $\equiv_M^k$ same as $\equiv_M$, but only looking at strings of length $\leq k$.

Observation 1: The equiv relations $\equiv_M^K$ satisfy: $p \equiv_M^0 q$ iff p, q both accepting or both non-accepting for $k \geq 1$: $p \equiv_M^K q$ ¡=¿ $p \equiv_M^{K-1} q$ AND $(\forall a \in \Sigma)$ $(\delta(p, a) \equiv_M^K -1\delta(q, a))$

Proof: See notes
Corollary: Can compute $\equiv_M^K$ for any k by dynamic programming in time $O(k|Q|^2|\Sigma|)$
Observation 2: Enough to continue until $k = |Q| - 1$ because then $\equiv_M^K = \equiv_M$
Proof: Start with 2 equivalence classes. At each level, either $\equiv_M^K = \equiv_M^K$ or increase no. of classes by at least 1
Can't have more than $|Q|$ classes, so recursion stops after $\leq |Q| - 1$ steps
When it stops, $\equiv_M^K = \equiv_M$, because $p_M q$ then $p_M^{K'} q$ for some finite k' and $\equiv_M^{K'} = \equiv_M^K$ because we stopped. $\square$

Algorithm:
Given DFA $M = (Q, \Sigma, \delta, q_0, F)$:
(1) Starting from $\equiv_M^0$, compute each $\equiv_M^K$ for $k = 1, 2, \cdots$ until no progress
(2) Define new DFA M' with states Q' = equiv. classes of $\equiv_M^K$. $q_0' = [q_0]$, $F' =_{q \in F} [q]$, $\delta'([q], a) = [\delta(q, a)]$ consistent by construction
(3) Remove any states that are unreachable from q_o'
(4) Output resulting M'
Running Time: $O(|Q|^3|\Sigma|)$
Can improve to $O(|Q||\Sigma|log(|Q||\Sigma|))$

Example:
$\equiv_M^0$: A, D, B, C, E
$\equiv_M^1$: A, D, B, E, C
$\equiv_M^2 = \equiv_M^1$

$\equiv_M$ states
$\equiv_L$ strings

8.2 Note 1

Q1: Given a regular language, how can we construct a DFA for L what has minimum number of states? Q2: Given a non-regular language, how can we prove the L is not regular?

Distinguishable Strings Def 1: $x, y \in \Sigma*$ are distinguishable w.r.t. L (a language over alphabet Σ) if $\exists$ $z \in \Sigma*$ s.t. $xz \in L$ AND $yz \not{i}nL$ or vice versa.
Two strings in are distinguishable if there is a string that allows append to both of then, but only one of them remains in the language.
The strings become indistinguishable if neither of them remain in the language.

Myhill-Nerode Theorem
State Minimization

9 Tuesday February 9th, 2021: Lecture 7

9.1 Live Lecture: First Pass

Logistics:
HW3 Due Friday at 5pm
Move from Monday to Thursday Lecture
Today's Topics:
Two highlights of FA
(1) Knuth-Morris-Pratt algo for pattern matching
(2) Firing Squad Problem
Start streaming algorithms

Pattern Matching
Input: Text of length n over alphabet Σ
Pattern y of length m on same alphabet
Output: Find first match of y as a contiguous substring of x
Assume: $m << n$

E.g. $\Sigma = A, C, T, G$
$n = 100M\ 10^2 7$
$m = 2^8$

Naive Algorithm: $O(nm)$
e.g. $x = AAAA....AB$, $y = AAB$
Goal: $O(nm)$

E.g. Text: $x = ABACCABABABACA$
Pattern: $y = ABACA$
KMP: Skipped matching, that uses information about the structure of the pattern
Q: How do we implement this idea?
A: Use a DFA!

FA Construction: Have set of states for each prefix of pattern y
Start state is empty string, and accepting state is entire string
You need to go state by state to figure out what the next best step is, according to a mismatch or match
The goal is to maximize overlap

q:= 0, i:=1

repeat:
q ¡- $\delta(q, x_i)$
i ¡- i + 1
until $q = m$ or $i = n + 1$
if $q = m$ then output "Match found at position $i - m$"
else output "no match found"

$O(n)$ + time to construct DFA

Constructing the DFA (Pre-processing step)
States: $0, 1, \cdots, m$
Transitions: $\delta(q, a)$ = maxj: y[j] is a suffix of y[q] a
where y[j] is the prefix consisting of first j symbols of y
Naive computation of δ
$O(m|\Sigma|mm)$ iterate over q, iterate over a, positions for prefix match, checking match
$O(m^3)$, so overall time is $O(n + m^3)$
Cleverer construction algorithm for DFA takes time $O(m|\Sigma|) = O(m)$
Overall running time $O(n + m)$ KMP

2 Other pattern matching algorithms
(1) Boyer-Moore Algorithm which is also $O(n + m)$
(2) Karp-Rabin randomized algorithm also $O(n + m)$, but with small probability of reporting false matching
Instead of matching patter, match fingerprints
Fingerprint of $y = ymodp$ where p is prime, so fingerprint $O(logm)$
Therefore, runtime $O(nlogm)$

Break

Firing Squad Problem
Linear array of n identical FAs (nn parameter)
At time $t = 0$:
-General is in state q_init
-Soldiers are in state q_sleep

Goal:
At some later time t, all the DFAs must enter state q_fire for the first time
Operation:
State of each automaton at time $t + 1$ depends on its own state and those of it's two neighbors at time t, i.e. $\delta : Q^3 \rightarrow Q$

FAs must work for any value of n
Idea: Divide and conquer
Find middle soldier
To find middle soldier:
-general sends out two "signals", one traveling at unit speed, the other traveling at $\frac{1}{3}$ speed
-the soldier who receives both signals at the same time is the middle soldier

Running Time $O(n)$
Optimal: $2n - 2$ time steps
Generalizes to "Firing Mob Problem" where soldiers are arranged in a general bounded degree network - time $O(d)$ where $d =$ diameter of network

Interview Question:
Input: Stream of n items over alphabet Σ, s.t. one element occurs $> \frac{n}{2}$ times
Goal: Find this element using only one counter of $O(logn)$ bits using only one pass through the stream

9.2 Note 2: Patterm Matching

Definition: string x, text of length n, string y pattern of length $m << n$, over alphabet Σ
Obvious Algorithm: Slides y along x and checks if substrings match at that index, running $O(mn)$

Knuth-Morris-Pratt pattern matching algorithm (1970s):
Complexity: $O(n + m)$
Example: $x = AbACCAbABACA$, $y = ABACA$
Once it finds it first mismatch, it jumps ahead to letter after first mismatch to continue
Sometimes, the new letter is not the start letter and it assumes letters already processed
First: Convert strings x, y into DFA

DFA Construction:
$0, \cdots, m$ states
Initial state: 0
Accepting state: m
Spine(transitions from i to $i + 1$): correspond to pattern y
All other transitions correspond to mismatched symbol
Takes to earlier state that matches longest matched prefix ending in current position of

text

Formal Construction:
Substring:
$y[k] = k$ length prefix of string y, $0 \leq k \leq m$
Transition Function:
$\delta(k, a) = maxj : y[j] is a suffix of y[k]a$ for $0 \leq k \leq m$ and $a \in \Sigma$
Algorithm:
$q := 0; i := 1$
$repeat$
$q := \delta(q, x_i)$
$i := i + 1$
until $q = m$ or $i = n + 1$
if $q = m$ then output "match found at position $i - m$"
else output "no match found"

DFA construction complexity: $O(m^3|\Sigma|)$

Other Algorithms: Boyer-Moore, Karp-Rabin

10 Thursday February 11th, 2021: Lecture 8

10.1 Live Lecture: 1st Pass

Missed the first 5 minutes from microphone
Today: Streaming Algorithms (Note 3)
Next Time: Context-free languages and Pushdown automata
Midterm Febraury 25th

Streaming Algorithms
There is a connection between streaming algos and finite automata
Stream of data $x_1, ..., x_n$
The data is likely large and not stored
Finite automata is a streaming algorithm with constant-size memory
Finite automata can't count number of zeros in string
Goal: What can be done with a small amount of memory of order log n $O(logn)$ where stream length is n
If we gave the machine linear memory, then it would be able to memorize the entire stream.

Example
$L = (0, 1)$ string with equal number of 0s and 1s
Algorithm: Maintain a single counter that records number of [1's seen so for] - [0's seen so far]
Counter $\in [-n, n]$
No. of bits $O(logn)$
Accept iff counter = 0 at end of stream

Use distinguishable strings for lower bound of strings.
Prove: It is not possible with less than $O(logn)$ memory
Def: For language $L \subseteq \Sigma*$, two streams $x, y \in \Sigma*$ are streaming indistinguishable for L on inputs of length n if $\exists z \in \Sigma*$ s.t. $|xz| = |yz| = n$ and exactly one of xz, yz belongs to L.

Reason for Definition (Theorem): Suppose $\exists$ set D(n) of pairwise distinguishable strings for L on inputs of length n. Then any streaming algorithm that recognizes L must use at least $log_2|D(n)|$ bits of memory on streams of length n.

Proof: Supppose for contradiction that $\exists$ streaming algorithm that uses m(n) ¡ $log_2|D(n)|$ bits of memory on inputs of length n. Then algorithm has at most $2^m(n)$ internal states of the algorithm which is less than $|D(n)|$. So two distinguishable strings $x, y \in D(n)$

that take the algorithm to the same state. But then xz, yz both in L fo both not in L.

Corollary: Need at least $|log_2(\frac{n}{2}) + 1|$ bits to recognize L = above.

Proof: Set D(n) = $0^i1^{\frac{n}{2}-i} : 0 \leq i \leq \frac{n}{2}$. $|D(n)| = \frac{n}{2}+1$.How do we distinguish $x = 0^i1^{\frac{n}{2}-i}$ and $y = 0^j1^{\frac{n}{2}-j}$. Set z = $0^{\frac{n}{2}-i}1^i$. Then $xz \in L$ and $yz \in L$. .

Interview Question: Finding a Majority Element
Σ: Large alphabet
Given a stream $x_1, x_2, ..., x_n \in \Sigma*$.
Assume $\exists$ majority element i.e., some $x \in \Sigma$ that occurs $> \frac{n}{2}$ times.
Goal: Find this majority element using only $O(logn) + O(log|\Sigma|)$ bits.
Trivial with $O(|\Sigma|logn)$ bits - maintain a counter for every element.

Break from 11:45-11:50
Maintain: One active element (or guess), and maintain one counter. Intially: counter = 0, no active element
Suppose: next element is x_i
Update Rule: If counter is equal to zero, make x_i active and set counter to 1
Else: if x_i is active, then counter = counter + 1
Else: Decrement counter by 1

Claim: If x is majority element, then it is active at the end of the stream.
Proof: Assume x occurs $m > \frac{n}{2]}$ times. Assume for contradiction x is not active at the end. For analysis, think of counter as a stack of occurrences of items. Associate every occurence of x_i with a distinct occurrence of same other elt. Assume for contradiction because there exists fewer than m such occurences.

Consider any occurences of x.
(i) Supposes decrements counter: Associate with occurrence of some x' that's popped off stack.
(ii) Suppose increments counter: Associate with occurence of sme x' that later causes this occurace. A x to be popped off stack.

What if x is a most frequent element but it's not a majority.
Assume l = $log_2|\Sigma|$ and $2^l > n^2$, i.e. $|\sigma|$ grows like n^2. We can now prove strong lower bound to recognize this language.

Claim: Assuming $2^l > n^2$, any streaming algorithm for most free element problem mst use at least $\Omega(nl)$ bits of memory on inputs of length n.

Proof: Work with simpler language recognition problem. Write $\Sigma = 0, 1, ..., 2^l - 1$. $L_{MFE} = x \in \Sigma* : most - free element is 0$. Goal: Find $2^{\Omega(nl)}$ distinguishabl strings for inputs of length n w.r.t. L_{MFE}. For each subset $A \subseteq \Sigma$ 0 of size $\frac{n-5}{2}$, define stream $x_A = 0, 0, 0, a_1, a_2, ..., a_{\frac{n-2}{2}}, a_{\frac{n-5}{2}}$ where A is X$_A without the duplicates.$

Family of distinguishable string will be $X_A : A subset of above..$
(i) Check that x_A, x_B are distinguishable $(A \neq B)$.
Let c be some element in $A\ B$. Let $z = c, c$.
$X_A Z : majority element is c (4 times)$
$X_B S$same as above
(ii) How many dist. strings ...
...

More generally, think of "inputs of size n" in a less rigid way.
e.g. testing whether a graph $G = (V, E)$ is connected.
n = no. of vertices.
G is presented as stream of edges $(v_1, v_2), (v_3, v_4), ...$
For lower bounds, find distinguishable sreams w.r.t. graph on n vertices (stream down't have to have length exactly n).

Big Picture:
Finite Automata = Regular Expressions
PDA (FA + StacK) = Context-Free Grammars
Read: Sipset 2.1 and 2.2
$A \rightarrow wBx$
Turing Machines (FA + General Memory)

10.2 Note 3: Streaming Algos

11 Tuesday, February 16th, 2021: Lecture 9

11.1 Live Lecture: 1st Pass

Logistics
Midterm Next Thursday
Overview
Past Topics
Today Topics
-Context-free Lanuages, Context-free Grammars, Pushdown Automatas

Grammars
Way of decribing languages using rewrite rules (production rules)
"—" OR symbol that combines many rules
Example
Digits, Ints, Signed$_in, Id, Letter, Val, or Arithmetic Expression$
$Parse tree doesn't show the order of derivations.$

Definition: Context-free Grammar (CFG)
Tuple $G = (V, \Sigma, S, R)$
Variables, Terminals (Alphabet), Start Variable, and Production Rules
Production Rule: Maps variable to string or variables and terminals
A given varaible can be on the LHS mapping to many rules
$A \to W_1|W_2|W_3...$
Starting from S (start variable), apply rules in any order to any variables.
Do this until left with $w \in \Sigma*$
G generates a string w
$L(G) = w \in \Sigma* : G(w)$, where $G(w)$ means G generates w

CFG sufficient to define programming languages
EXAMPLE 1:
Allows for program to compile sufficiently.

EXAMPLE 2:
Balanced Parentheses over $\Sigma = (,)$
$S \to (S)|SS|\epsilon$
((()())
Note: Languages of balanced parentheses is not regular
$REGCFL$

EXAMPLE 3: Equal Number of O's and 1's

$S \to OA|1B|\epsilon$
$A \to 1S|0BB$
$B \to 0S|1AA$

EXAMPLE 4: Pailindromes
$PAL = s \in 0, 1* : w = w^R$
$S \to 0S0|1S1|0|1|\epsilon$

CFLs: languages generated by CFGs

Machine Model: Pushdown Automaton
We want a program that checks if a string is accepted, useful for Compilers
$w_1, \cdots, w_n$ with a pointer from Q
Includes stack (unbounded memory LIFO) $s_1, s_2 \cdots$
After Q changes, stack added or popped off

PDA is an NFA augmented with a stack
(1) reads next input symbol (or ϵ) and reads or pops symbols on top of stack
(2) enters a new state and writenew symbol

Note: always nondeterministic
Formally: $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
$\delta : Qx(\Sigma x\epsilon)x(\Gamma \cup \epsilon) \to P(Qx(\Gamma \cup \epsilon))$
$L(M) =$ Language accepted by M (non-deterministic)
Mapping from $q \to q'$ where mapping is $a, \delta \to \delta'$
$(q', \delta') \in \delta(q, a, \gamma)$

EXAMPLES
(EX 1) $L = w \in 0, 1^n$: was has some no. of 0s and 1s
repeat, compare next input symbol with top of stack
if stack empoty or symbols watch thm push symbol back down
else: pop stack

Until end of input
Accept iff stack is empty

Deterministic
(EX 2) PAL
(1) [Phase 1] Read input symbols and push onto stack
(2) Non-deterministically guess middle of input and more to Phase 2

(3) [Phase 2] read input symbols, compare will top of stack and POP from stack - die if they don't match
(4) Accept iff reach end of input on empty stack
(5) Die if anything bad happens

Break: Resume at 12:10
Non-Deterministic (unavoidably)
...

Theorem: A Language L is a CFL $\iff$ L is recognized by a PDA
Proof:
(i) CFL =¿ PDA
Given a CFG $G = (V, \Sigma, S, R)$, construct a PDA as follows:
(1) Push start symbol onto stack
(2) Repeat until end of input:
(2a) If top of stack is a variable A, non-det. choose a rule with LHS A and replace A on stack by the RHS of rule.
(3) If top of stack is a terminal $a \in \Sigma$, then read next input symbol - if it is not a, then die, else pop a and continue

EXAMPLE:
$S \to (S)|SS|\epsilon$
Input: (()())
$S \to (S) \to S) \to SS) \to (S)S) \to S)S) \to)S) \to S) \to (S)) \to S)) \to)) \to) \to$
Plug and chug rules to get the form
This sequence ends up with an empty stack

Proof:
(ii) Given a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
Construct a CFG, G s.t. $L(G) = L(M)$
Idea; introduce a variable A_pq for each pair of states $p, q \in \mathbb{Q}$. A_pq will generate all input strings that take M from state p to state q on empty stack
Note: Must start and end with empty stack
Important: This allows for you to compose machines together if stack is empty

Then we'll be done because A_{q_0,q_f} (where q_0 start state, q_f accepting state - assumed unique - of M) will generate exactly $L(M)$ -assuming also w.l.o.g. that M empties its stack before accepting

Grammar G
$V = [A_pq : p, q \in \mathbb{Q}$ where $S = A_{q_0,q_f}$

Production rules
(1) $\forall p, q, r, s \in \mathbb{Q}$, $\delta \in \Gamma$, $a, b \in \Sigma \cup \epsilon$ if $(r, \gamma) \in \delta(p, q, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, \gamma)$ then include rule: $A_p q \rightarrow a A_{rs} b$
(2) $\forall p, q, r \in \mathbb{Q}$, include rule $A_p q \rightarrow A_p r A_r q$
(3) $\forall$p $\in \mathbb{Q}$, include rule $A_{pp} \rightarrow \epsilon$

Induction proof verifies correctness of this grammar: i.e. prove by induction on no. of steps in computation: $\forall p, q \in A_p q$ generates x =¿ x can take M from state p to state q on empty stack

Final Words: Closure properties, parsers, deterministic CFLs
Note: We have enough information to finish homework

NOTE: This lecture is impressionistic instead of technical details

Context-sensitive Grammar, also has rewrite rules, but only able to apply them when variable in a specific environment.

Read up on Wiki Page

11.2 Textbook Notes: Sipser 2.1

We previously studied finite automata and regular expressions.
Context-free grammars describe describe recursive structures (i.e. found in natural languages with nouns, verbs, and prepositions).
Used in compilers, where parses are designed after CFL specified.
Context-free languages = languages associated with context-free grammars.
Includes all regular languages and additional languages.
Pushdown automata - machines that recognize and provide insight into CFLs

Terminology:
CFG is a set of substitution rules (productions).
Rule maps a variable to a string of varaibles and terminals.
There is a start variable.

Algorithm to Determine Language from Grammar:
(1) Write down start variable
(2) Replace a written variable with RHS of matching rule

(3) Repeat till no variables remain.

A feasible sequence of substitutions is called a derivation that can depicted as a string or as a parse tree.
All possible strings make up the language of the grammar and and language made using CFG is a CFL
The english language can be outlined with a CFG.

Formal Defintion
Tuple (V, Σ, R, S)
(1) V is finite set of variables.
(2) Σ is finite set, disjoint of V, called terminals
(3) R Finite set of rules, where each rule is made of a variable and string of variables and terminals.
(4) S is start variable s.t. $S \in V$.

11.3 Alistair's OH

Decision problems are easier to get results on.
Learning Automatons

12 Thursday, February 17th, 2021: Lecture 10

12.1 Live Leecture: 1st Pass

Logistics
HW4 Due Tomorrow at 5pm
No HW next week
Midterm next Thursday 2/25 during lecture shart
Look out for Midterm document this weekend

Topics:
TODAY:
Finish CFLs, Deterministic CFLs
Start Turing Machines
LAST TIME:
(1) Context-free grammars
e.g. $S \to (S)|SS|\epsilon$
$S \to SS \to (S)S \to ((S))S \to ((S))() \to (())()$

$L(G)$:= set of strings generated by the grammar starting from S
(2) Pushdown automata
Non-det FA + stack
Move depends on state, input symbol and top of stack symbol
move changes state and potentially pops stack and pushes new symbol onto stack
Acceptance is as for non-determinism FA
L(M) := set of strings accepted by M
Theorem: L is a CFL ¡=¿ L is recognized by a PDA

Properties of CFLs
(1) $REGCFL$ e.g. balanced parentheses and palindroms $\in CFL$ but $\notin REG$
(2) But: CFL's are also limited. e.g. $0^n1^n2^n : n \geq 0$ not a CFL
e.g. $ww : w \in 0, 1*$ is not a CFL
e.g. $ww^R : w \in 0, 1*$ is a CFL
(3) CFLs are closed under:
union $S \to S_1|S_2$ where S_1 is start symbol of state of grammar 1 and S_2 for G_2
concatenation: $S \to S_1S_2$
Star: $S \to \epsilon|SS_1$
(4) CFLs are not closed under complement or intersection
e.g. $a^nb^nc^nn, m \geq 0 \cap a^mb^mc^n : n, m \geq 0 = a^nb^nc^m : n \geq 0 \notin CFL$
hence not closed under intersection. Also not closed under complement (since we can express intersection in terms of complement and union)
(5) By definition CFLs are nondeterministic. Restrict to deterministic PDAs (DPDAs) $\to$ get smaller class of languges
e.g. PAL $\in CFL\ DCFL$
DCFLs are not closed under union and intersection

concatenation: $S \to S_1S_2$
Star: $S \to \epsilon|SS_1$
(4) CFLs are not closed under complement or intersection
e.g. $a^nb^nc^nn, m \geq 0 \cap a^mb^mc^n : n, m \geq 0 = a^nb^nc^m : n \geq 0 \notin CFL$
hence not closed under intersection. Also not closed under complement (since we can express intersection in terms of complement and union)
(5) By definition CFLs are nondeterministic. Restrict to deterministic PDAs (DPDAs) $\to$ get smaller class of languges
e.g. PAL $\in CFL\ DCFL$
DCFLs are not closed under union and intersection

concatenation: $S \to S_1S_2$
Star: $S \to \epsilon|SS_1$
(4) CFLs are not closed under complement or intersection
e.g. $a^nb^nc^nn, m \geq 0 \cap a^mb^mc^n : n, m \geq 0 = a^nb^nc^m : n \geq 0 \notin CFL$

hence not closed under intersection. Also not closed under complement (since we can express intersection in terms of complement and union)
(5) By definition CFLs are nondeterministic. Restrict to deterministic PDAs (DPDAs) $\rightarrow$ get smaller class of languges
e.g. PAL $\in CFL\ DCFL$
DCFLs are not closed under union and intersection

...

Got sleepy, I will go back and update this and the rest of the notes by Wednesday in preparation for the midterm and upload it accordingly.

12.2 Textbook Notes: Sipser 2.2, 3.1

PUSHDOWN AUTOMATA
NFAs + Stack
Stack provides additional memory aside from finite amount in control
PDA can recognize nonregular languages with help of stack
Writing symbol is called pushing and referring is called popping from top of stack
Stack can hold infinite amount of memory
Therefore PDA can store numbers of unbounded size
There are deterministic and non-deterministic pushdown automatons

Finite automata and pushdown automata are too restricted to serve as models of general purpose computers.

TURing MACHINES
Finite automaton with unlimited and unrestricted memory
Problems unsolvable by turing machines are beyonnd theoretical limits of computation.
MEMORY: Infinite tape, with tape head that can read and write symbols by moving on tape
PROPERTY: Continues computing until produced output of accept or reject

13 Tuesday, February 23rd, 2021: Lecture 11

13.1 Live Lecture

(1) HW - none this week, HW5 out on Friday
(2) MT - 2/25 starts 11am sharp, read instruction on Google doc on Piazza, many ques-

tions, brief, concise but precise answer, read questions first, don't worry if you don't finish, figure out how to do the problem solving before hand

TODAY:
-Continue with Turing Machines
-Church-Turing Thesis
-Reading -Chapter 3 Sipser, Note 4 (turing's 1936 paper) and Note 5 (RAM Model)

Turing Machine Model:
Q - controller
Tape - Fixed left and infinite right
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_acc, q_rej)$
M = (states, input alphabet, tape of alphabet (Γ), ($b \in \Gamma$), state, accepting, rejectin)
Transition function: $\delta : (Q\{q_acc, q_rejx\Gamma) - > Qx\Gamma xL, R$
Deterministic by default
Configuration of TM: is of form uqv starting substring, controller, ending substring
On given input $w \in \Sigma*$, TM M may:
-accept and halt
-reject and halt
-not terminate ¡- not accepting

Definition: A language L is Turing-recognizable, recursively enurmerable, if L is the set of strings accepted by some TM
Definition: A language L is Turning-decidable, recursive if L is the set of strings accepted by a TM that halts on all inputs
Tuples: (Read, Write, Move)

Example:
See Book

Break 11:48-11:52

Turing Machine
Bells and Whitles
(1) Two-way infinite tape
(1a) Initial head position $\cdots, s_{-2}, s_{-1}, s_0, s_1, s_2 \cdots$
Claim: 2-way and 1-way infinite TMs are equivalent
1-way infinite simulation of 2-way infinite TM, be "folding tape"
$\Gamma' = [s, t] : s, t \in Gamma \cup [s, \$], s \in \Gamma$
$\Sigma' = [s,] : s \in \Sigma$

$' = [,]$ $Q' = [q, u], [q, D] : q \in Q \cup q_0, q_acc, q_rej$

(2) Multiple Tapes
(2a) k tapes has k heads
$\delta : Qx\Gamma^k \rightarrow Qx\Gamma^k xL, R^k$
Initially: Inpute on tape 1, all other tapes blank, all heads at left-hand ends
Note: Time to simulate T moves of k-tape volume is $O(T^2)$

E.G. Palindromes can be recognized in $O(n)$ time on a 2-tape machine, require $\Omega(n^2)$ time on a 1-tape machine

(3) Multidimensional tape
contains storage tape and work tape

(4) Nondeterministic TM
Claim: Non-deterministic Tms are equivalent to deterministic TMs
Proof: Simulation of a nondeterministic TM by a 3 tape deterministic TM
Input, Simulation, and Control tape

Repeat:
-Copy input on tape 1 to tape 2 (erasing everything else on tape 2)
-Use tape 2 to deterministically simulate M on input w, looking up nondet. choices on control tape
-If no more choices on control tape, or if choice is invalide, abort this ram
-If simulation accepts then accept
-Replace control string on control tape by lexicographically next string
-Loop

13.2 Note 4

Algorithm - effective procedure
Models aims to formalize notion of effice procedure - e.g. Turing Machine
Q: Can all efficetive procedures be described by Turing machines?
Q: Can java describe process not capable by Turing Machine? (i.e. TM is too simple)

Church-Turing Thesis: Any process called effictive procedure can be realized by Turing machine
Parallel formalism exists in λ-calculus

Thesis not theorem, because ”effective procedure” is inutiutive notion and not capable under formalization

13.3 Textbook Notes: Sipser 3.1

Theorem. *test*

TM cannot solve certain problems

These problems are beyond the theoretical limits of computation

Memory: Infinite tape

(1) read, write tape operations (2) head can move both L and R (3) infinite tape (4) accept and reject states take effect immediately
Definition. *Turing Machine:* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

The alphabet Σ *includes blank symbol*

$\delta : Q \times \Gamma \to Q \times \Gamma \times L, R$

Def: Configuration: current state, current tape contents, and current head location

Def: yields: C_1 yields C_2 if the machine can legally transition between the two configs in a single step

Def: TM M accepts input w if (1) C_1 is start config of M on input w (2) each C_i yields C_{i+1} (3) C_k is accepting config

Def: Collection of strings that M accept is language of M, L(M)

Def: Language is turing-recognizable if some Turing machine recognizes it

Def: TM M can (1) accept (2) reject (3) loop

Def: Loop means never leads to halting state

Def: TM M is decider if only (1) accept or (2) reject

Def: A decider that recognizes some language *decides* that language

Def: A language is *Turing-decidable* if some M decides it

Covered four examples of informal descriptions converted to formal descriptions

Example 3.7, 3.9, 3.11, 3.12

14 Thursday, February 23rd, 2021: Midterm 1

14.1 Prep

Review Discussion 1 to 5
D1 1.6, 1.41, 1.31, 1.32 D2 1.60, 1.31, 1.48, 1.21, 1.22, 1.67 D3 1.53, 1.71, 1.58, 1.72 D4 1a, 1b, 2, 3a, 3b, 4a, 4b, 4c D5 2.18, 2.44, 2.19, 2.23, 2.24

Textbook Q's w/ A for Chap 1 and 2

14.2 D1

Important Defs:
Definition of DFA:
$M = Q, \Sigma, \delta, q_0, F =$¿ Sets of states, alphabet, transition function, starting state, set of ending states.
Note: Every state of DFA has exactly one exiting transition arrow for each symbol of the alphabet.
Definition of NFA: Each state may have zero, one, or many existing arrow for each alphabet symbol.
Zero, one, or more ϵ transitions may exist as well
NFA splits into multiple copies of itself, when action is taken with multiple arrows
If any copy of NFA is in accepting state at end of input, string is accepted
Closure Proofs
Closure of Union: (1) $Q = Q_1 x Q_2$ (2) $\Sigma = \Sigma_1 \cup \Sigma_2$ (3) $\delta((r_1, r_2), a) = (\delta(r_1, a), \delta(r_2, a))$ (4) (q_1, q_2) (5) $F = (F_1 x Q_2) \cup (Q_1 x F_2)$

14.3 D2

14.4 D5

Important Defs:
Definition of CFL:
Built from CFG, and modeled using ND-PDA
Def of ND-PDA:
$M = Q, \Sigma, \Gamma, \delta, q_0, F$
$\delta : Q x \Sigma_\epsilon x \Gamma_\epsilon \rightarrow P(Q x \Gamma_\epsilon)$

14.5 Score and Thoughts

Score: 10/52 Median: 32

15 Tuesday, March 2nd, 2021: Lecture 13

Logistics: Midterm 1 has been graded, and you should spend at least one hour to go over the sample solutions. Also note, that regrades are open from Tuesday midnight to Friday midnight. Homework 5 is due Friday at 5pm. Also, move to Thursday's discussion from Monday due to load balancing.

Topics for Today: Church-Turing Thesis: RAM Model, Undecidability of Halting Problem

Church-Turing Thesis: Any effective procedure can be realized by a Turing machine. It is a thesis, and not a theorem, because it cannot be proved, but it more of a philosophical statement. Therefore, any device can be modeled by a Turning machine. However, things that are based on arbitrarily small accuracy, we can't model them using a Turing machine.

Evidence in favor:

(1) All other attempts to formalize computation have been shown to be equivalent to TM (e.g. λ-calculus found in functional programming (Church, 1941), production systems (Post, 1941), recursive functions (Kleene, 1952), Type-0 grammar (Chomsky, 1959)). These are used to model computation and can be equivalent to Turing machine.

(2) Random-access Machine (Cook, Rechow, 1973)

Goal: Cover equivalence of Turing Machine and Random Access Machine

RAM: Has memory made up of (1) Memory (registers) (2) Input (finite stream) (3) Program

Configuration at any given time is $s : \mathbb{Z} \to \mathbb{Z}$ where $s(i)$ = contents of registers. The RAM starts off with empty registers and at any given time there will only be finite data, so it's a finite function. We also need a program counter to tell us which line of code to execute (i.e. next line of code) and input pointer, which tells where to read the finite input.

Initially: s is the 0-function, the counter is the first line of code, and the input pointer is at the start of the input. RAM can both accept, reject, and loop infinitely.

Note, all real world problems can be represented as a decision problem.

RAM defined as a context-free grammar, with optional strings noted by '[]'

Example: Represent Palindromes as RAM

(1) Simulating a RAM by a TM

w.l.o.g. we can use a multi-tape: (1) input tape (2) storage tape (3) work tapes

Input tape has fixed input size, and storage tape is designated by a $ symbol. Storage tape is a representation of the registers. Key question, how do we store ram register on storage tape. We organize them as $a_1 : v_1 a_2 : v_2$

$\cdots a_m : v_m bb \ldots$, which means that value v_i is stored in register a_i. So, if register a is assigned a new value v_1, just append $a : v$ to the end of the storage tape. To loop up value in register a: scan storage tape-right-to-left; value s found in first record containing a $(a : v)$. If no such record is found, then the value is zero.

To simulate RAM, we have a block of states for each RAM command in the program. e.g. read L by reading next symbol from input, then evaluate L using look-up and work tapes, then assign input value to register that L evaluates to

If using assignment command $L := R_1 \circ R_2$, then we evaluate R, L_1, L_2 using lookup and work tapes, then assign value $< R_1 \circ R_2 >$ to register $< L >$ if $R_1 \circ R_2$ go to λ, then evaluate $R_1, R_2, R_1 R_2$, then transfer control to appropriate next block of states dependent on boolean value of $< R_1 \circ R_2 >$

(2) Simulating a TM by a 3-register RAM

$a_{-1}, a_0, \ldots, a_k\,(q)\,, a_{k+1} \ldots b, b, \ldots$. Now Let $m = |\Gamma| + 1$ where Γ is the tape alphabet. Given each symbol an integer code in $0, 1, \ldots m - 2$ where blank gets symbol 0. Assign left marker a_{-1} the code m-1. Then use three registers $-1, 0, 1$.

Implement a move of the TM:

-current symbol is s (register 0)

-current state is implicitly stored in current block of code

-e.g. assume left move, and overwrite register 0 with new symbol, and replace r by r*m adds to r, replace s by l-m*(l div m),

NOTE: I switched taking notes to a graphic tablet, which I will review to then type up into something coherent in LaTeX after reviewing the notes.