CS 181 Notes

W 1 M Lec 9-26-16

0. Deterministic Finite Automata
   - Most fundamental notion in the theory of computation

1.1 Basic notions

Def. An alphabet is any finite set of symbols.

- We take a bunch of symbols (doesn't matter what they are)

Def. When you join symbols, you get words.
A string is any finite sequence of symbols from a given alphabet

- abc001 is NOT a valid string over the English alphabet or the mathematical alphabet, but it IS valid for a combined English/mathematical alphabet

Def. a language is a bunch of strings over some alphabet.

- A set of strings over some alphabet

- Unary alphabet consists of only one string repeated several times
- Does this mean there is only one language over that alphabet?
- No! You can choose the length of the words over the alphabet.
- An alphabet can contain stuff like 00 instead of just 0s and 1s
- This arbitrariness makes us prime to interpret visually some written symbols
- A person from a different culture will interpret them as different symbols
- Mathematics is blind to these cultural distinctions

Finite: There is a limit to the number of words that fits a language rule
Infinite: There is NO limit to the number of words that fits a language rule

Def: A computational device is any mechanism that inputs a string and produces an answer. ("accept" or "reject")

- We are going to figure out what computation is.
- The definition must be very abstract in general.
- There used to be a device called a lightswitch with two states (on or off)
- It takes this input and produces an output (turns the light on or off)
- The number of computational devices is very large
- You take a leaf on a plant and the way that surface tension works in the morning is the output
- Many more examples!
- Repeat the experiment and get additional bits of the answer

1.2 Deterministic finite automata (pl.)

automaton (sing.)

- Distill what is important and then you make your definition

Rules:

- There has to be a finite number of states
- There has to be AT LEAST one state

- Choose your start state
- Choose accept states
- Draw a transition (from every state on every symbol)

- You must show every transition possible and account for every possibility

Def. The language of a DFA (deterministic finite automata) is the set of strings it accepts

1.3 Designing DFAs

W 1 W Lec 9-28-16
2 Nondeterminism

2.1 Basic notions

- There isn't a whole lot you can do with an automaton; you need to introduce additional features to make it more capable
- Nondeterminism is a fundamental feature that arises every step of the way as you study the theory of computation.
- One of the most important questions in all of mathematics and computation
- P = NP -> Does nondeterminism add any computational capability

Bolded is the same for DFAs!

- **Choose your alphabet {a, b}**
- **Draw your states**
- **Choose your start state and accept states**
- Draw transitions

- There are sets of unhanded states

NFA accepts w iff <u>at least one</u> path leads to an accept state

Epsilon as a string is different from epsilon as a transition

- Epsilon transition is a transition that you take any time you aren't willing to use any of your actual characters
- Freebie you can always take no matter what your characters are
- Epsilon transition means you are always free to take it or free to ignore it
- Stops at some point and then it starts to match the pattern

- The pattern it can match is 101 or else 11. It then gobbles up the remainder of the string
- The language is the set of all strings containing 101 or 11

2.2 Using shortcut

- You need at least one state as long as it suffices
- You don't have to designate all the states and transitions
- You either end up in a rejection state or you blow up (don't need to handle errors)
- **As long as there is one path that leads to acceptance, you are good!**
- Use middle state to gobble up any intervening characters

Q. Why would you want this model because it leads to unhanded exceptions?
A. The beauty of non-determinism is that it is very easy to build machines in this model. Produce very succinct designs for complicated behaviors. Deterministic analog would be a tremendous hard work

Q. Is every non-deterministic NFA has a deterministic equivalent?
A. Lead you to believe that NFA are these magical machines that have this awesome power and it is very unlikely that a deterministic machine could ever compete.

2.3 Pattern matching

- You must recognize precisely what the problem asks for!

2.4 Alternatives
Ex. {w: |w| divisible by 2 or 3}

Don't overoptimize your constructions on exam

W 1 Dis 9-30-16
Pei

W 2 M Lec 10-3-16

- Compare DFAs and NFAs and see that in some ways, they are equivalent and in some ways, they are NOT equivalent
- Fundamental philosophical implications

2.5 Formal Definitions

- You tell it what state you are currently in (Q) and then you tell it what character you would like to use for your transition (or epsilon).
- Then the machine tells you what your options are in the Power set

Diagrams are useful for reasoning about a problem, but as a way of mathematical communication, it is NOT preferred.

Q. How do we distinguish between empty set that cause machine to blow up vs empty sets that stay in the same place?
A. It is only an issue if the transition is an epsilon transition. If there is an empty set, how do we know if it is a true, empty bonafide empty set?

- It is a hard problem algorithmically and it isn't clear how to do it.
- You can try to change any given entry and see if that change produces a functional effect.
- There isn't a nice, short characterization to answer that question.

Q. If there is an empty set on epsilon, is it like receiving an epsilon?
A. When do you receive an epsilon, there has to be AT LEAST 1 path of acceptance. If you inadvertently introduce some epsilons, that may lead you to blow up.

Def. NFA acceptance

- NFA accepts a string if there is a set of paths that lead to acceptance
- We can intersperse a bunch of epsilons and make sure that the string spells **w** (language)
- Otherwise, it is NOT a legal computational path on input **w**
- There needs to be a computational path such that the sequence of states you traverse works
- The next move should be one of the legal next moves and so on and so forth.
- The final state q_m should be a legal possibility of where you end up, AND it has to be an accept state

- **an accept state is reachable from q_0 via some path on w**

0. (Non)Equivalence of DFAs + NFAs

- DFA automaton cannot represent as a single accept state (accepts all strings) or a single reject state (rejects all strings)
- You need at least 2 states

3.1 Example

- We started with this simple looking question and we solved it using a precise, formal procedure whether deciding if a string is accepted by a given NFA
- We can see where we end up from there using a new set of possibilities and we can keep going until we have examined all of the chunks of the string
- Computational procedure
- Computer program can do this!
- Can a DFA do it?

A DFA can be equivalent to this!

- As long as your set of states includes an accept state, you are good!

Q. How come there could be states that cannot be reached?
A. You can further simplify this automaton by removing states that cannot be reached.

3.2 General Theorem
The. For every NFA N, there is a DFA D that recognizes the same language.

- What is the return value of this function?
- A subset of Q
- If we had a subset of the power set, this would violate the return type.
- A script F is whatever collections that are good for us.
- Functionally it is the same

Q. The return type of delta, is the return type part of the power set or the original set of states?
A. It is an element of the power set, so any subset of Q

W 2 W Lec 10-5-16
- You blow up in size when you go from NFA to DFA
- There are languages that inherently require DFAs even if they have a small NFA
- Def. A language L is called <u>regular</u> if some NFA recognizes it
- Equivalently, a language L is <u>regular</u> if some DFA recognizes it
- To prove if it is regular, exhibit an NFA for it because you have all these shortcuts and additional features you can use to simplify the design

3.3 Blowup in size
L has an NFA with k states => L has a DFA with 2^k states

Pf: Guess a prefix of some length and then gobble it up.
- Remainder of the string should have exactly k characters, the first of which is a 0
- We need to rule them out without expecting them individually
- Take a much more general result which will be the culmination of our study of automata

- Mathematicians have a claim that is morally correct
- Why is something like this supposed to be true?
- For DFA, it has to have a state for every single suffix
- The states are what the automaton uses to memorize things
- Stores intermediate information
- A lot of information that it needs to remember, and we cannot keep track of this information with a small number of states

- After it reads the whole string in entirety, let's say it ends in q_25
- In reality, we are almost finished.
- **The key observation is that these cannot all be distinct!**

Run the two experiments simultaneously.

- Originally, both strings hover over the start state and let's press play.
- They start meandering all over the place and we are running the two experience in parallel
- Eventually, they will converge and the automaton no longer remembers the two strings it dealt with
- Once the two strings converge, you cannot tease them apart.
- The only memory of the past experience is whatever it has stored in that past state.
- You cannot distinguish between u and v at that point
- You will always end up at that same state

Q. Why are we using i-1 instead of k-1?
A. This string when you look at it closely has a 0 from the kth position at the end. When you work out the details, it doesn't come out that way

- An automaton is memoryless in that it's only knowledge of previous symbols is in its current state
- Pigeonhole principle: Lots of strings on the left but a small number of states
- There aren't enough states to go around.
- There will inevitably be some collision for the same state that will result from at least two strings.

- As far as u and v are concerned, it may well be possible that the automaton gets them both right.
  - At least one of them will be wrong, and the string v with a bunch of 0s appended at the end
  - At least on one of them, it has to be mess up.
  - The first one has a 0 from the kth position from the end, and the second one has a 1 from the kth position from the end

- When you look at the logic of the proof, it goes like this.
- Take any DFA, and let's fix a DFA.
- The DFA can be arbitrary as long as it has fewer than 2^k states
- Once you have fixed it, we are going to show that it will have two strings where it gets the answer wrong.
- For every DFA that is small, there exists two strings where it messes up on at least one of them.
- Rules out a small DFA recognizing the language.

Q. If our input is all possible 2^k states, why can we say one of them doesn't exist in the language?
A. The strings have to differ in some bit position. Let's say in the ith bit, all we need is to know that the first string has a 0 in the ith position, and the other has a 1
- They cannot have the same value at that index.

- They have to disagree in some index and the other has to have another value (opposite value).

- It doesn't matter what you call them (u and v), but mathematics does NOT distinguish between them.

0. Closure
- Important in mathematics

4.1 Complement
- Can you recognize the Union or Intersection of these two languages?

- Requires so much creativity
- Q. Have we seen an example of a language that is NOT regular?
- A. We haven't and we have NOT shown that yet.

If you would like to develop some intuition for this, try to build an automaton {0}+ followed by the same number of 1's
- Same number of 0s as the number of 1's
- You cannot do this with an automaton but we cannot show it yet.

- NFA is a bit more tricky, but if you change the accept and reject states, it can mess up your pattern matching.
- It starts to behave completely unpredictably.
- Construct a concrete automaton would lead to disaster
- The proof doesn't go through if we replace this with an NFA. This doesn't mean that the claim is false, but the same line of reasoning is no longer valid here.
- A statement can be true even though the justification provided is false.
- This just means we haven't tried hard enough.

Q. How would we leverage this to our advantage?
- By working with the complement instead of the original language, you can simplify your job.
- Work with the complement of the language first.

- A DFA is already an NFA, but it chooses not to use the additional features of Non-determinism

4.2 Union + Intersection
- Draw a cloud
- When you think you have the right idea, that's when you turn to notation to formally write it down
- How do we combine them?
- Draw another state and combine the two!
- You need to think creatively and abstractly in terms of symbols
- Use DeMorgan's Law
- Build on theorems we learned before

- Worked hard to build it, so let’s take advantage of it

- Build on stuff that you or other people built previously
- Take the complement via Theorem 1 and use the rule from (a)

- How do you convert this intersection into an automaton
- Cartesian product construction: He just kept track of where is fingers were
- The automaton just had to keep track of a single finger, but now it memorizes where two fingers are.
- It is all the same to the automaton

- DFA is more cumbersome but it is extremely robust!
- NFA is simpler but easier to make a mistake

W 3 M Lec 10-10-16
- You have to really think outside the box
- Highly sophisticated results

3.3 Difference.
- All strings in the first language that are NOT elements of the second language
- L” is the set of things you cannot have in your license plate (swear words or offensive words)
- Does NOT contain an offensive letter combination
- L” is the set of all swear words
- Your license plate should NOT contain it for obvious reasons

L’ is a regular language and L” is a regular language: Keep only strings in L’ that do NOT have a substring in L”
- Set of L’ is the set of all valid license plates
- Set of L” is the set of all swear words

3.4 Concatenation
- Cross product
- Any language concatenated with empty is empty

- If you start to approach the problem trying to chain some closure properties, you will likely NOT succeed
- Check the first automaton and see if it reaches an accept state. From that point, it should have the option of teleporting to the 2nd automaton
- By withdrawing the accept status, you are making them transition and NOT follow up with a substring on the second language
- You make these reject states and then you add this epsilon transition on top of all the transitions you already have.
- Feel free to transcribe it formally!

- For a concatenation, remove the accept state of the first condition and add an epsilon transition to the start state of the second condition

3.5 Kleene Star

- Mathematical induction is a theorem.
- If the base case is true and the inductive step is true, any **finite** number of steps will be true, but we cannot apply this to **infinite** and this will have to assume additional things about this system.
- Morally correct solution: All the correct ideas but tweak the implementation slightly
  - Add extra state and then proceed with the same idea

- If the start state is zero copies, we have the accept state at the beginning, then we take an epsilon transition to the normal start state. We would then reset and and reach another accept state possibility and we are NOT committed to the strings in the language.
- If I want three copies, we just reset three times.

3.6 Prefix & Suffix

- Prefix: Whatever stage that leads to acceptance, mark as acceptance!
- Requires a profound understanding of the automaton model to arrive at it
- Suffix: Sometimes, reverse is covered before prefix and suffix, but assuming we don’t know about reverse, how would we handle this?
- Epsilon arrows to all states reachable from q_0

3.7 Reverse

Def. reverse(L)

- What will the final character be?
- It is just unthinkable that the reverse of a language should be regular as well
- For it to be a general theorem, it would be very surprising
- Someone gives you a string, and asks you to verify whether the reverse of that string is in a language.
- How would you even attempt a problem like this?
- We need a conceptual breakthrough to completely shift gears and bring a completely different mindset to this question
- What if you flip all the errors but you have to readjust to the new initial states
- BRILLIANT!

By walking backwards, you would have reached the starting state from an accept state!

- Alternate problem imposed under suffix.

3.8 Substring

3.9 Shuffle

- Think about this problem just like all the other problems as an engineer
- Imagine you are the automaton and someone gives you the string
- Separate it out
- One liner (the problem is one of the hardest in this unit)
- End of the first unit of CS 181
- All the material on the first exam comes to this.

W 3 W Lec 10-12-16

- Figure out where these breakpoints should be placed.
- Shuffling
- Each transition has its own context, so move only one hand at a time.
- The logical way to think about this is that you can only move one piece at a time, so the hand movements of the 1st machine and the 2nd machine should both be accepted for the shuffle to be accepted
- Some law that you follow for your finger movements.
- When your fingers at certain states, what are your legal moves.
- If sigma is part of L_1, then you use the transition function from L_1, otherwise you use the transition function from L_2 if sigma is part of L_2

What if you cross two sets? You get a set of pairs.

- What if you replace this by a comma, this would be problematic!
- States live in the plane and don't live on the real line.
- One dimension is the first finger and the other dimension is the 2nd figure.
- We cannot have a state with just one component.
- If you describe your solution clearly enough in words, it is enough!
- You must convey your solution clearly

- Test questions require an in-depth understanding of the model but don't have long answers
- This machine does NOT enforce alternation!
- The language doesn't enforce alternation because these are strings, so any of these can be empty.
- If a particular AI is empty, this means we took a pass on this automaton on that particular step.
- These chunks can have zero length.
- In fact, you are allowed fewer cross-hatched chunks than the unshaded chunks.

- The new language is obtained by getting the old string in the language and plugging it in at either the beginning or the end.
- Could you separate it into prefix and suffix and have a new state recognize it from the language.
- Consider a prefix of the language. Concatenate it with the alphabet, and there is NO way to enforce that the prefix comes from the same string as the suffix

- You have a choice of what problems you solve on the exam
- 25 points worth of problems (perfect score is 20, so you get to pick which 20 points to solve).
- Give you flexibility to shop around for questions and it gives you the possibility of making partial progress on every question and you can get a perfect score on every part.

5 Regular Expressions
5.1 Basic Notions
- Ways to describe languages additively
- You add things and at no point are you allowed to subtract.
- Those operations are destructive and can result in the removal of characters.
- Inductive definition != Circular definition
- Circular definition begs the question
- Inductive definition bottoms out so that is the ground objects.

Regular expressions are memoryless like automata
- Sigma Sigma stands for any two characters over the alphabet
- Sigma is infinitely stretchy and absorbs those
- Regular expressions are additive, you can NOT remove stuff from the language

Described in different language but the two NFAs are actually the same but labeled differently.

Q. Given two regexes, is there a way to prove they recognize the same thing for all strings?
A. A final exam problem! Highly motivated problem because a TA gets several answers and wants to check if they are the same as the official answer.
- Plug this into a computer that says equivalent or not equivalent.
- Is there a way of checking of infinitude of possibilities?
- There is a way and after today's lecture, we are in good position to do it.
- Don't repost copyrighted material
- Click the link for the discussion section and please don't repost them because we won't have any leverage left with the publisher.

Regex and automata are exactly equivalent
- Allows us to think of regular languages in a completely new light.
- NFAs can be expressed as a regular expression
- About pattern matching and regular expressions
- 5.2 Equivalence
- If something is represented by a regular language, how do we know that there is an automaton for it?
- Regular languages are closed by the removal of a regular language

- There is no such capability with regular expressions
- You cannot intersect two regular expressions
- Thus, it is surprising that regular expressions should be closed under the idea of intersection
- Think of an automaton (2D object and the nodes are laid out in some complicated fashion)
- Arrows from one node to another
- Very complicated!
- Regular expression is a linear, 1D object
- How could you take a graph and compress it to something 1D?
- This was considered very surprising and a celebrated result of elementary automata theory
- By ripping out a state, we need to repair the automaton so we will add arrows to the automaton to patch things up and account for the removal of this state.
- Then, we will pick another state and patch it up with arrows.
- 9

W 4 M Lec 10-17-16

- Pick the state where you have to do the least work
- Quite arduous because there are two donors and three receivers in the example for B
- The end state will be the same. Just make it easier to get to that final state!

0. Pumping Lemma

- Challenging because you need to throw out every conceivable attempt at this problem
- Show there does NOT exist any automata forever.
- Size of up to 6 or 7 won't be verified to recognize this language.
- Need a conceptual breakthrough that let you argue that we can prove this is non-regular
- If you wanted to recognize n 0's and n 1's, you needed n memory, so you cannot have infinite states to recognize
- Somehow, the automaton has to remember how many 0's it has seen so far, and whatever memory it needs has to memorize the path
- Only vehicle of memory are the states of the automaton.
- n can be arbitrarily large, so make sure your automaton remembers how many 0's it has seen
- Why is it that before he reaches the accept state, he must have visited a place more than once.
- We know for a fact that we are going to end up at an accept state, so at every step of the way, why cannot we be somewhere new?
- Read p symbols and we have already visited a state
- At some point, I will just run out of states because I have p steps to go, and only p-1 possibilities for new destinations

Every regular language has this pumping property

- To prove that a language is not regular, it suffices to prove that it does NOT obey that law.
- Here is our language of interest and therefore, it is NOT regular.
- Use the lemma's contrapositive form
- If it rains, I get soaked: statement A
- If I don't get soaked, then it hasn't rained -> contrapositive of statement A

If it's winter => no leaves on the trees: Original statement B

- Swap the premise with the conclusion and negate both

If there are leaves on the trees, it is NOT winter: Contrapositive of B

- Write the language as a complement of the previous language

W 4 W Lec 10-19-16

- The quantifiers are very powerful and you don't want to interfere with them
- You cannot make any assumptions about p
- On the HW, someone gives you the concrete value, and once someone announces that number, we are in charge all of a sudden
- Pick a string in that language of at least that number length
- From then, we have no power again since the adversary partitions it into 3 chunks (x, y, z)
- We cannot make any assumptions!
- In whatever way the adversary pleases, the adversary will split it up in the most convenient, adversary manner
- There are actual partitions s.t. y is NOT empty, and $|x| + |y| <= p$
- You have to prove that whatever the decomposition is, it cannot be pumped.

Prime number

- Think of a prime number and add delta to it
- You might hit another prime number, and if you keep adding delta to it
- No infinite progression consisting of prime numbers

Fractional part of pi starts with w

- y can be infinite, since we did not prove infinite is regular

Exam preparation:

Understand all the examples we did in class

- Know proofs

Understand the homework solutions

- The homework is really important to doing well in this class
- Graded based on effort so that you seek help and collaborate
- Talk to them for the homework and look up solutions on the Internet as long as you are learning

Solve the practice exams
- Do additional problems from the textbook

You cannot beat practice!
- Just do it!
- Roll up your sleeves and just do it!

You just need a pen or pencil
- No scratch paper, notes, cheatsheets, or any of that crap

Perfect score on the exam is 20 points, but there are 25 points worth of problems
- You get to shop around and pick what 20 point worth of problems you want to invest your time in.
  - Focus your energy on a subset of problems and do them correctly
  - Show partial progress on every problem

Why does regular - non regular have to make regular?

6.5 True or False?
- “Regular langs are closed under countable unions”
- “Nonregular langs are closed under complement”

Most exam questions will be short and sweet, but requires you to think

Q. Does the pumping lemma always work? If a language is non-regular, will some application of the pumping lemma get the job done?
A. No, unfortunately. Some languages are NOT regular, but you cannot use the pumping lemma to prove it.

- Exhaustive list of strings

W 5 M Lec 10-24-16
- Started new unit on Myhil-Nerode
- Pumping lemma proves for a wide range of languages that they are non-regular, but some things are off-limits for the pumping lemma.
- So elegant that once you see it, that’s the whole idea and you will forget the pumping lemma
- Usually the wording will not specify which method you want to use, so you can use a pumping lemma or Myhil-Nerode
- There may be problems that explicitly ask for proofs using the pumping lemma or Myhil-Nerode
- More complicated models will be only use the pumping lemma as the method of choice.
- Regardless of the current situation, it will come in handy.
- There is a universal tool called the Myhil-Nerode theorem

Def: When you tack w onto x and tack w onto y, either both strings are in the language, or neither strings are in the language

- Automatons classify strings into accepted or rejected.
- The automaton would like to get that job as inexpensively as possible using as few states as possible.
- Do I need a separate state for x and a separate state for y?
- Do we need to see if it is just x or y?
- If they are indistinguishable, it does NOT matter if you have just seen x or just seen y.
- Whatever continuation we followed, we cannot tease x and y apart.
- It doesn't matter if we have seen x or y.
- It suffices to remember we have seen one of them.

Whether two strings are the same or not the same depends on the language in question

- Whenever your relation is an equivalence relation, it simplifies things dramatically

- Strings are always distinguishable, so therefore, if you were to shape the regions of the circle.

Atomicity: A class is entirely in L or entirely in !

Tack on 1 for both cases but 001 will be rejected automatically

Let's say you have two strings 00110 and 0

- Tack on a w at the end of both strings s.t. in the 1st case, the first would be in the language and the second would NOT be in the language
- In this case, try adding 1 to the end
- 001101 is NOT in the language, but 01 is in the language

Number of classes == number of states

7.3 Myhill-Nerode Thm

- Cycle through a small number of states and get information about the language.

- Most instructors prefer NOT to teach the Myhill-Nerode because it has a reputation of being very challenging, but once you wrap your mind around it, it makes things easier!
- Clarifies a lot of things and helps you understand what a regular language is.

- Look at the various equivalence classes and be concrete.
- Pretend that the first string you see is the empty string (epsilon)

- When you see a symbol, you do NOT memorize it!
- Look at the map here and there is a new symbol called sigma that lands in the 3rd region, so pretend that I have just read the string s3.
- What happens here?
- Compressing!
- The string that I have seen so far is s3 and it is false so the actual string I have seen could be very different.
- Look at your map and see where does this new string land on the map
- Pretend it is the region s7 and pretend that it is quite different from everything else.
- I will arrive at exactly the same conclusion that I would arrive at if I were to memorize the entire string (entire billion symbols)

Transitions: s_i, sigma

- Where in the map does that land?
- The region where it lands is the string that I am going to pretend I have seen
- For the next time period, I will have seen where to go.

- Cycle through the equivalence classes according to this map
- The minimum size of a DFA is exactly equal to the number of equivalence classes
- Two applications of the Myhill-Nerode Theorem are proving non-regularity of non-regular languages or proving the optimality of your DFA.
- The Myhill-Nerode Theorem has very important applications to that language.

Compare i 0's with j 0's.

W 5 W Lec   10-26-16
7.4 Proving optimality

- Exhibit 4 strings s.t. any two of them can be teased apart.
- Any two of them can be in different equivalence classes

This is a tricky example so we automatically force one of them out of the language and it may happen that a match occurs not where we expected.

- We expect it to occur at position 0 and it could occur later on in the string

End of Exam 2 Material

There are many languages that do not get recognized by this model

- Take it up a notch and study a more sophisticated model.
- We are now going to see that even this model is not powerful enough to handle everything, but we can change it ever so slightly and have the power of universal computation.

- Evolution is extremely rapid and we have an intermediate model and the next step is everything.

Intermediate model is very natural and it is a great model that is motivated by two applications

- Natural languages (spoken by humans)
- Programming languages (parsing theory)

8 Context Free Grammars

- Let’s specify an alphabet and a set of variables

First Exam Results

- A’s - 55%
- B’s - 24%
- C and Below - 21%

Look at your exam and if you notice any inconsistencies, come talk to them.

- Regrades are not bugging, but on the contrary, it takes a strong person to come to the instructor and say there’s a misunderstanding
- TA is the first point of contact and they are far more qualified to talk about the exams

We could hold office hours on that Monday and we can look at the exams

- We need your careful look at those exams and figure out stuff.
- You’ve done well on the exam, so maintain the momentum.
- Exam 2 is harder than Exam 1, and Exam 3 is the killer exam
- Exam 4 (Final) has been much easier than Exam 3
- Try to keep up the good work and use your resources when you need it.
- Abundant office hours and face time.
- Do NOT fall behind in this course!
- **For those who didn’t do as well, try to do better and make use of resources.**
- Help everybody get A’s and B’s in this class
- No adversarial curving that enforces that there is an upper bound in A’s and B’s

- Takes some creative thinking to verify this, but this is a really good example

8.3 Constructing a grammar

- If you wanted to generate an excess of apples, you can invoke this rule

W 6 W Lec 11-2-16

- Ambiguity is a big thing
- This problem is really tough and it asks for an unambiguous grammar
- Can you come up with a grammar period?

- That is very non-trivial
- There is a grammar for this language that uses a single start variable

- Simple != easy to produce
- Understand it easily but coming up with it is very different!
- To solve this problem, you have to think like an engineer
- Look at first prefix that is a string in L
- Excess of a's
- Break this down into substrings!
- String in the language and either we can break it down into a concatenation of two strings or else we will have one such string

Start off with an a, end up with a b
- This cross-hatch will have equally many a's and b's, and the remainder will be S again
- The problem asks for something even harder
- Come up with a grammar that is unambiguous
- This grammar is extremely ambiguous!
- We cannot enforce the fact that we are considering the shortest prefix with equally many a's and b's

We need a conceptual breakthrough and a completely new way of thinking
- To solve the original problem, we need something more powerful!
- The idea is to graph your string! (WTF)

If you graph starts at 0 and ends at 0, your graph has equal a's and b's
- What if your graph never dips below the x-axis, no prefix has more b's than a's

For someone to give you a grammar and decide is one of undecidable problems of nature.

- Break down hard problems to smaller ones that are simpler
- What substructure exists for strings in this language?
- Parentheses!
- Handle strings where the prefix has the same # of a's and b's

- There is a prefix where there are exactly as many a's as b's

- Come up with a grammar that is unambiguous, and these grammar problems are more challenging than what you would see on an exam
- When you graduate and are asked to do computational tasks, it would actually come up!
- Facebook forces you to disambiguate grammars! (Badass internship)
- Context-free grammars are the hardest part of this course!

Automata are easier to program and we understand them better.

- There is a type of automaton that corresponds to context-free grammars.
- There is an automaton that precisely corresponds to regular expressions

9 Pushdown Automata

9.1 Definition

- Have arbitrarily many items on your stack

- If there are still problems you cannot solve with this model, but if we drew a 2nd stack, that would be it so we would have the powerful of universal computation!
- A modern computer corresponds to an automaton with 2 stacks!
- Once you have 2 stacks, you can do anything!
- Anything after 2 is unnecessary, 2 is the magic number!

- You can think of writing on tape, which is arbitrarily as long as you like.
- Tape corresponds o scratch space.
- You can write things to disk and look at that in detail later on.
- Add this new feature to a model that we understand well.

Top of the stack should have A, otherwise we cannot transition!

- PDA's are very succinct and we can represent advanced behavior.
- A lot of power in this model

A few words about Exam 2

- This dramatically reduces the amount of stress you have to face because you get tested on a small amount of material
- Exam is 1 hr 30 min, and generally, you will need that length of time to solve every problem if you want your solutions to be very detailed.
- Very compartmentalized and try to reduced the inner dependencies on each exams
- Exam covers stuff since Exam 1, which is regular expressions, non regularity (pumping lemma and Myhill-Nerode)
  - Context-free grammars are not on Exam 2

- Exam 2 is noticeably harder than Exam 1
- It took Pei half an hour to do Exam 2
- More challenging if that is any indications
- On Exam 2, questions dealing with P.L. or Myhill-Nerode are must-haves!
- Do NOT lose these points!
- On any exams, there are non-standard questions and you have to pray.
- Pumping lemma and Myhill-Nerode have to be rock solid!
- If you want to keep these points, study your ass off.

Historically, most people end up losing a lot of points on those problems

- Do NOT apply P.L. on a constant length string!

- Please make sure you study at least those points so it is automatic.
- Problems on the exam where you are really off the beaten path and we have less control of that situation.
- Be sure to get those and regular expressions!
- Go into a cycle and establish a routine because this promotes a reduction in stress.

- You need to somehow memorize the # of a's, but you cannot use the scratchpad because that is unbounded memory
- Take advantage of stack!
- Anytime you see a, write it on a plate and put it on
- At some point, from now on, I will be processing p's
- Now I am in a different state of mind and if I see a b, I pop a plate from my stack and discard the plate.
- Once your stack empties, then you accept.
- Now, we get in this loop where we see an a in the input string and don't pop anything on the stack.
- See another a and keep pushing it onto the stack
- Stop non-deterministically.
- We make this hop which requires reading nothing from the string and pop nothing and push nothing.
- This transition is always applicable.
- We see a b and pop a from the stack and what happens if you pop an a and there isn't an a on the stack.
- You blow up and this is only usable

In the end you don't need to pop the bottom marker! To go here, you can just erase this and make it epsilon!
- How would you implement this language with even fewer states?
- I want a two state solution and both of them are accepting.
- You never need to clean up before you accept.
- Whether you accept or not depends on if your current state is accepting or rejecting.
- At the start state, you just push a's and the second one, you pop a's

Programming a PDA is way easier than coming up with grammars
- Mike Sipser in the official solutions writes that the problems I assigned are the hardest in the textbook.
- Gamma can be the same as sigma and it can extend sigma in some weird ways or extend empty

Delta has a domain and a range:
- It yields a power set, and the ease of implementing various languages skyrocketed!
- It became a much more capable model.
- It is really helpful!

- Adopt a human-centric view of the process.
- Clearly practice this at home, and visualize a stream of a trillion symbols in arbitrary order and all you have is this little notepad where you can jot down notes, but you have finite # of pages
- You have this stack of plates and you can remove a plate from the stack.
- See what computational behaviors you can realize in this model.
- You really understand this model and become highly proficient at it.
- Going this formal route gets you nowhere.

- The diagram is going to take care of itself, and it is what happens at the end of the solution.
- The solution is the creative process, and the diagram is just a transcription of that process.
- This is just a mechanical process described without an automaton.
- You read the symbols of the strings one by one, write it on a plate and put it on the stack.
- Write it on the plate and it will be a very tall stack.
- At some point, we decide that the time is right.
- How do we decide? You never know and we feel that the time is right.
- You do something quite different, and we see a symbol from the input and we make sure that the input is the same thing.
- We need to make sure what is on the plate matches what you read.

9.2 Constructing PDAs

- Implementing this is a matter of mechanics, so the hard work is behind us.
- This is the icing on the cake if you will.
- Not only does this hop occur at a non-deterministically chosen point in time, but we can decide whether to consume a symbol from the string.

- Think in terms of the scratchpad and this stack of plates and at some point, you switch to taking off the plates and see if you end up at the bottom.
- We need to store u somehow on the stack.
- Push the letters on to the stack, but when you pop it, at some point you pop an a and b at the same time.
  - You are storing u, the first string, on the stack.
  - Put one on top of the other and now at some point, you decide that you have read u.
  - You have a stack that is a trillion characters tall and now, someone gives you v.
  - Match v to u but the first character of u is at the bottom and you have no access to it.

- When you have a third plate, you need to stick them under the first two.
- One stack only!
- Once you have two stacks, you can do anything!

- This is fundamental to being a computer scientist, but this is not why Sherstov does it.
- He does it because it is fun and it is awesome.
- Creating something out of thin air and Sherstov really challenges to think of a stack of plates and a little scratchpad, and see if these strings are distinct.
- You cannot take your stack of plates and flip it, and you cannot reach for the bottommost plate in your stack without removing everything.
- Do not think in terms of the automaton model but rather a bio mechanical process.

W 7 W O.H. Sherstov 11-9-16
- All strings in language of length up to n the grammar generates them.
- Prove that all length of n+1 can also be generated by the grammar, and argue by induction on the length of the string.

2.27
- The compiler pairs your else with the closest if statement that precedes it
- We are going to mimic this behavior, and whenever we pair else with if, it matches the closest one by default

Deciding whether a grammar is ambiguous or not is impossible.
- There is not a systematic procedure or algorithm to decide ambiguity
- Undecidable problems in nature.
- Whenever we are faced with a problem, we have to throw all our tools at it and hope we can solve it.

Context-free: Language that can be recognized by a context-free grammar or a PDA
- Regular languages correspond to regular expressions, NFAs, or DFAs

More powerful model of context-free grammars.

Queue is worth two stacks: a queue is much more valuable than a stack

Disambiguating a grammar is undecidable
- You have to reason on a case by case basis
- No recipe and just a few tricks that you learn by practicing

W 7 W Lec 11-9-16
9.5 Summary
- Context-free language can be recognized by a CFG or a PDA
- If it is regular, it has an NFA or DFA. An NFA or DFA is a PDA that doesn’t use its stack

10 Non Context Free Languages
10.1 Pumping Lemma
- Pumping behavior is basically the same but you have strings v and y

- If you pump them in simultaneously as many times as you like, the string will be in the language.
- Regular languages - pump one string
- Pumping lemma - pump two strings simultaneously
- Proof in the concept-free case is so much more intuitive and cleaner than the proof for regular languages
- More coherent somehow.

Complex analysis vs real analysis

- When you start to study complex analysis, it makes so much more sense and things feel right.
- The right way of looking at things, things work predictably.
- Real analysis is a total mess
- Folks usually find it intimidating, but there is no reason to find it intimidating but it should be easier to understand

On the next exam, anything having to do pumping lemma should be **guaranteed points** because it is very predictable and no surprises

- No control over the decomposition, but all bets are off.
- These three properties!
- If we are really unlucky, we will just have 3 strings and pump it regular language style.
- The adversary may decide to come up with two. It is the same skill, so if you have practiced the pumping lemma for the previous case, it should be no different!

Repeat a loop and still stay in the language!

- Why doesn't this hold in the context-free case?
- Consider a very long string much longer than the strings in the automata
- At some point, you are going to get to a state where you have been before.
- This gives you a loop and can we just continue the argument or is there something different?
- **The stack will be different!**
- With PDAs, your snapshot at any given point isn't just your state of mind! It is also the contents of the stack.
- Perhaps on the first revolution of the loop, there are no guarantees that when you reach the same state for the second time that the stack would look the same.
- Things aren't quite the same because you haven't stepped into the same river twice.
- You have to reason very differently!
- Our string is long but it is finite, so we have to take the perspective of a context free grammar.

Get to define what long enough means

- Let b = max length of r.h.s of any rule in our grammar in R
- Look at all the rules in our grammar and see how wide the right-hand side is.
- The r.h.s is whatever is to the right of the arrow.

Any string with **b** characters can be pumped!

- There will be some pair of repeated variables i.e. A and A

Unhinge the outermost tree and suspend it into this pivot.

- Won't even notice anything because the string will e what it was at the beginning
- The tree will be smaller, so we assume that the tree is the **<u>smallest</u>** possible!

Proof is elegant and uses elementary data structures like **trees**

- The proof is much shorter and cleaner

Exam 2 results

- 33% A's, 31% B's
- The results came in late last night
- Thankful for the TA's to grade around the clock since the exams were administered

A: 18-25
B: 16-17.99
C: 14-15.99

- Sizeable group of folks who got perfect 20 on this exam because the material was difficult
- People who struggled a lot on Exam 1 and worked so hard that they scored a lot higher than on the previous exam

Exam 3 is historically the most challenging one and it is psychological

- GRE is very important for grad school
- Hard prerequisite for getting into a good graduate program
- Sherstov wasn't nervous and scored 2390 out of 2400 on the GRE!

Context-free involves two substrings and together, they are NOT wider than p characters including the portion in between them.

10.3 Examples
<u>Ex</u>:

- Make sure number of a's and number of b's is the same
- You have lost the n for a's because you popped things off the stack
- Now you will rigorously prove this language is NOT context free.

- There are languages that fulfill the criterion for context-free that are NOT context-free!
- No other recourse in this model!
- The pumping lemma is our only tool here!
- This is why the pumping lemma is still taught for regular languages, but now it is NOT the case since it is our only tool.

- We are in charge and can make p any arbitrary value we want!
- Afterwards, the adversary is now in control
- Good practice on the exam to explain rationale for using pumping lemma on context-free grammars

0. List all the possibilities to have a complete solution
- From that point on, the adversary can split it up, but from there, we have control again s.t. the result is NOT in the language.

W 8 M O.H. (Sherstov) 11-14-16
- Shorthand notation for PDA
- As long as it conveys information clearly, that is fine.
- If the TA cannot understand what you had in mind and misinterpret it, then that is bad. Otherwise, feel free to use any notation you are comfortable with.

2.20
- Use cross product construction
- Move finger around in B and see if you can reach an accept state
- You start by taking a walk in this automaton, and we decide we are done reading the string.
- At that point, we switch gears and start moving in lockstep.
- If they both accept, then you are done.
- If they are both regular, you can use a cross product construction, but you need a second copy of construction A
- The first language is context free, so that is what uses the stack.
- Use it during the first phase and the second phase.
- To use a PDA, you have to use a stack since it is part of the automaton

W 8 M Lec 11-14-16
- Closure properties really require to synthesize everything about the computational model
- Start with simple theorems and lead to questions

11.1 Union + Concatenations
- Do the same things we did with regular languages!
- Either use the PDA view of the model or the grammar view of the model!
- For the union, the PDA view is very short and elegant
- For the concatenation, it is problematic
- If you take the grammar POV, these are both short statements!

- Take rules of first grammar and rules of second grammar.
- Branch to the first symbol of first grammar or first symbol of second grammar
- How to change this to get concatenation of the two languages?

11.2 Kleene Star

- Typically, people give a slightly different solution first, and that solution is generally incorrect.
- Use the same variables in addition to a new variable called S_new

11.3 Intersection + complement

- The grammars cannot really give much from the approach of intersection
- Pushdown automata for the first two languages, but this is NOT particularly effective!
- You need two stacks and each of them needs a stack of its own.
- Presumably, you can run them sequentially, one after the other, but how will you realize that?
- You have no input to work with at that point.

Not closed under context-free automaton means context-free language

- Need both stacks to run, and because you only have one stack, you cannot runt he comparison to this!
- If you don't need the stack, the the Cartesian Proudct is unnecessary

Gamma is our stack alphabet

11.4 Reverse

- Exam question in a previous year.
- If you have a mirror, hold it perpendicular to the left of your nose and look in your phone. That will read grammar of the reverse!
- Try and test method of taking a PDA and running it in reverse.
- A little bit more subtle and you have to worry about the stack.

11.5 Prefix + Suttix

- Prove original prefix(L) is context free
- Stretching from here to the moon, we have an input of billions of symbol
- We want to find if it is a prefix in the language
- Turn every state that has a path to an accept state into an accept state (This is for regular languages, not Context-Free Languages)
- If it is an accept state, we are done, and we should accept state.
- Check if there is a path from where we ended up at to an accept state
- PDA is a very fragile model and it isn't just about where you are at, but also what your stack plates are!
- If you arrive at accept state and there are some elements on the stack, it might blow up.

Solution

- Make a copy of the PDA. Follow epsilon transitions to new PDA machine and see if it reaches new accept states.
- Try to do it yourself and the other approach is to delegate
- When a task is too big for you, delegate and appeal to some other authority.
- Create a 2nd copy of the PDA and decide if it is possible to reach an accept state by reaching some arrows

- Stack is with you at all times at any state you go to.
- When you go for a run, you are comfortable for the first few minutes, but think about what happens next!
- When you are 30 minutes into your run or 40 minutes, it starts to feel really good.
- You want to keep going because it is amazingly liberating!

End of Exam 3 Material

Turing Machines

- The previous automaton was limited so we added a stack.
- Now with a stack, we have a program that gets interpreted as a parse tree!
- This uses a context-free grammar called Java or C++
- Programming languages and natural language processing requires context-free grammars
- There are languages such as ww, where w is an arbitrary string, that are NOT context free!

If you give a computational model 2 stacks, it can solve any problem in the universe.

- Took truly remarkable researchers working in computer science and math and it took almost a century to arrive at this conclusion.

read/write head: Can both read and write contents to a cell

- Start at leftmost cell, look at a particular symbol, and the transition function instructs you to update the state
- This shows anything that can be computed in the physical universe that humans occupy
- Long held belief in the scientific community
- When you start to investigate these attempts, it is quite mind-boggling and none of them have succeeded.

The worst thing when sending a resume to a company is NOT hearing back!

- Rejected explicitly or leaving you hanging
- In both cases, you don’t get the job!

- If they contact you and say you are accepted, then you are accepted!

Turing machines are what we have been programing all along (computers)

- Once you get comfortable with these definitions, this is the easiest model to work with

Imagine a stream of bits stretching indefinitely from where you are to the moon.

- All you have is a scratchpad that can hold a finite amount of info (your state of mind).
- You can also walk up and down this tape and you can travel forwards and backwards.
- At the end of the day, you have to enter the state q_accept if the input is in this format
- If not, all bets are off!
- You can write something on this plank of wood and imagine you are walking and the railroad tracks are so long that when you have walked far enough, you don't remember anymore where you are!
- It all looks the same in terms of railroad tracks
- This is what is so powerful about this analogy, and they extend indefinitely in both directions with no landmarks.
- All you see is the plank of wood where you are currently standing.

- At the first point, you memorize the symbol, leave an indicator, and then check the # and go back and forth.
- You can only memorize a finite amount of information

Questions to ask

0. How to do regular expression -> pushdown automaton questions?
0. Assumptions to make on Fall 2015 #4 if it says regular (can we assume that it is potentially context-free?)?

W 8 W Lec 11-16-16

- Mark as memorized and carry it with you to the other side of the # side
- Skip any X along the way and sure enough, you see your first one and mark it as checked
- Only X's to the left and right of the # sign and then, you do one last scan of the string to make sure you have no more characters unaccounted for

This solution is quite wasteful!

- You could memorize the next symbol and compare it to the 1st symbol on the left of the pound sign.
- For the sake of simplicity, do NOT worry about this!
- Illustrate how to implement this with a Turing machine.

Tape alphabet is completely arbitrary!

- It is a fixed number and can be arbitrarily large, but it has to be a fixed number.
- Make due with 8 states.
- Specify the transition diagram
- For every tuple, it provides an output, and you can enumerate this in tabular format. That would be rather lengthy though.
- More compact representation that makes a lot of sense.

Notations

- To the left of -> is before
- To the right of -> is after

No q_reject?

- This is NOT complete!
- Very compressed and assumes that the input is formatted as is and you don't encounter any unexpected behaviors.
- A string that doesn't have a # sign.
- What happens here?
- Branch at the beginning and skip any 0's and 1's and keep going until we see a # sign.
- We will hit a blank before we hit a # sign

- What if we were to get mixed up and get a 0 here?
- It is completely unnecessary!

12.3 Another Example

- Check whether the # of 0's in this sequence is a power of 2
- All you have is a scratchpad and a way of marking things up on the table.
- Shuttle back and forth, however you please.
- Once you have walked out enough, you have lost all landmarks and you are lost at sea!
- You better have a plan because all you can do is move forwards or backwards.
- You cannot memorize where you are!

Q. When you write something down, can it be an infinite set of symbols?
A. How much information can it hold? 16 GB, etc., but it is finite!

- Once you are out of space, you cannot write things on your phone. You must overwrite some old data.
- It could be a very expressive alphabet or a set of Chinese characters.
- A set with a million symbols but it has to be a fixed alphabet.

- In a particular cell, it has to be just one symbol because a cell only contains one symbol

Vast array of bits and it isn't like you can add things together to compute it.

- At some point, you run out of space in your head to keep track of this counting.
- Find halfway point by shuffling back and forth, and keep doing it until an odd number, or you hit 1.

- Someone walking on that railroad track and you can only mark things on the planks that support the tracks.
- Many solutions to this problem!

- When you discard half to the zeroes, you are preserving the invariant that the string has length of 2^p
- Keep dropping symbols until we have length 1!
- We expect to cross things off in pairs.
- On the first scan, encounter an odd number of 0's that is > 1.
- Whenever you scan your string, you have to see how man 0's you got!
- Make sure that it is an even number.

- Forward process where you are marching through the string and keeping track of how many 0's you have seen so far.
- Simultaneously check off every other zero

Backward process of this diagram?

- Top segment of the diagram

Eventually, we reach the end of the string which is the blank symbol, and then we know we need to rewind there!

- It could be s.t. we have no transition on a blank, and this is intentional because we blow up at the end.
- Step to the left and skip any intervening x's and 0's.
- Eventually, I will see the left-end of my string that I mark with a blank.
- Then, I do my next iteration and it will happen that the only 0 left is the 0 we marked with a blank character.
- When we reposition the head, the first symbol we hit will be the head (blank) and then we are going to accept.

Q. Is this the flipped version of the top?
A. Yeah! We check off the first 0 and mark it with a blank. This is where the tape starts and if we are referring to the blue chalk, it is for something else.

Next week's hw will ask you to implement simple behavior using Turing machines and it is a skill that will come very naturally to you.

- Important skill you have to pay attention to.
- You will then better understand what is allowed and what is NOT allowed!
- You may fall into a trap that Turing machine can do something else!

12.4 Yet another example

Leave markings on the tape because you cannot commit things to memory!

- A bunch of a’s then a bunch of b’s
- Check if number of a’s divides the number of b’s
- **For every a, check off a b!**
- If you run out of b’s and there are a’s remaining, then this means a’s do not divide b
- If you run out of a’s and there are still b’s remaining, restore the a’s and check off b for every a

Dot indicates location for the outer for loop

- We will also have an inner for loop

Turing machines have the exact same control features that we are familiar with

- Variables: How does a Turing machine implement a variable?
- It can only hold a fixed number of values
- Same for a computer: A computer cannot operate on arbitrarily large integers
- No hardware operation to multiply 2 arbitrarily large integers
- At home, when doing the next HW assignment, go through an implementation of one of these problems to create a full-blown transition diagram.
- Figure out what operations are legal and illegal!
- **If you were to explain your algorithm at a high level, an explanation like in CS 180 would be okay on an exam!**

People have been banging their heads against the wall to create more expressive models by social networks and no one has succeeded so far.

- Every extension to the Turing machine model has been showed to be equivalent to the bare bones machine!

13 Turing-Equivalent Models

- In 100 years, none of us will be around.
- Computing technology will be radically different and even more user-friendly
- We cannot even imagine how computers will be 100 years from now!

MacBook Air is not more capable than a TM

13.1 Bidrectional tape

- Mobius strip

13.2 Multiple tapes

- State of mind can only hold a finite information and we cannot memorize this!
- Simulate this array of k tapes with just a single head.

Questions to ask Pei

Q. Converting PDA to a grammar example
Q. Prefix CFL. Why do we only care about the stack?
A. Change this to epsilon to guess what to read. Go whenever is appropriate and take advantage of non-determinism

- Guess the suffix has sigma. The machine would pop gamma and pop gamma'.
- This path requires the stack contents to be the same because the transition includes the stack contents.

Hypothetically, your stack contents are quite messy, so you cannot satisfy the stack condition to get to the accept state

W 9 M Lec 11-21-16

- Provide evidence for the Church-Turing thesis
- No attempt other than the Turing machine has succeeded!
- People have proposed crazy devices and models, but each has been shown to be equivalent to the Turing Machine
- Very ad hoc and almost pulled out of a hat, but it is extremely robust and very universal

- Write symbols for each of these cells, and you then get to move each of the k heads independently

Thm: Adding k tapes doesn't do you any good, but 100 tapes can be simulated by a single tape.

- How would you simulate k tapes with a single tape?
- Erase these dividing lines?
- There is nothing wrong with it, but we will use that idea at the end of the day.
- By itself, it doesn't get the job done.
- Erase these lines and now you have bigger cells since every cell is a k tuple.
- The problem is that you still have just one head!
- You can interpret the k-tapes as a single tape by gluing them coordinate-wise.
- The problem is that at the end of the day, you have only one head
- One head has to simulate k-heads, and there is NO obvious way to do it since it needs to be at multiple places at the same time.
- These locations can be arbitrarily far apart.

- You cannot memorize that the first head is hovering over cell 32 and the other over cell 1024, etc.
- Not enough memory (finite memory)

- Let the tape do the work for you!

- Record the actual locations on this tape, or else, take it a step further!
- Mark the locations of the heads with circles and drop a pebble not he character where the head is on that particular tape.
- The tape does the job for you!

- Compute next move of the k-tape symbols
- Scan entire tape left to right, then scan backwards updating contents of the circled cells and shifting circles one at a time
  - Not a particularly fast algorithm O(n^2)
  - Same computational power!
- Commit the tape to memory.
- Store a finite amount of information in state and state q can store that information since the set of states you have is an arbitrary, finite state.
- Memorize a finite amount of information that is more or less the same as simpler models.

You cannot memorize the k-locations!

- You cannot memorize a number that has no bound on the number of digits it has.

13.3 Nondeterministic TM

- Let's run the Turing Machine and see what happens.
- Transition function specifies 2, 3, or 4 courses of action.
- Then, you cannot replicate yourself since you aren't an Operating System process!
- You are at a crossroads since there are 4 different ways you can go.
- You MUST try all 4 of them!
- There may be only one path of acceptance, so you have to try all 4 and see if at least one of them accepts.
- Each of these 4 tapes will encounter a point when there are more than one courses of action specified.
- Do they also split into multiple tapes?
- NO!
- At each point, arbitrarily choose a path, leave a marker, and try all solutions.

Project gets hijacked and you need get to it.

- Clearing out your desk!
- If I were to do it, it could easily take the rest of the day, so don't go down the rabbit hole!

What if you run the TM at each forking point and make a choice.

- First try out the legal sequence where you choose the first turn at every forking point.
- This may turn out to be a rabbit hole, so you may never terminate!

- First, try out all legal sequences of moves of length 1, then all legal sequences of moves of length 2, …
- Breadth-first, NOT depth-first!

- Non-Deterministic to begin with, but once you make a choice, it becomes completely deterministic!

Q. When we are doing this scenario, we have already done the scenario where we have 4 timed steps. It doesn't matter because we have to do it all over again from square one because we don't know how things look at the end of computation.
A. We have bounded the restrictions!

- Only run it for 4 steps in this case.
- If the Turing Machine accepts the input, there is a particular scenario that works!
- It takes some convincing, but this really works!
- How do you figure out the # of branches?
- Specifically, it is recognized by non-determinist machine.

Q. Do you only increment the last number on the scenario tape?
A. Up to you! Sherstov thinks of it as a calendar that checks if we reached a maximum value.
- If you can think of another scheme, by all means, there is nothing that is special about this.

13.4 Automata with multiple stacks
- Ties everything that we learned so far together
- Magic # is 2!
- You only need 2 stacks of plates!
- Think of the implications of this!
- Let's say you want 3 piles of plates, you can do it, but it won't have extra benefit.
- Using just 2 stacks you can get many capabilities.
- Take home message: To achieve general purpose computation, you need 2 stacks

k-taped Turing Machine is equivalent to a single taped Turing Machine
- We know that any k-taped Turing Machine can be converted at the end of the day to a single taped Turing Machine.

- Whenever you see an automaton with 57 stacks, you can see that it can be simulated by a Turing machine with a single tape!

Slinky - a good illustration of this model!
- Going to create a character (blank symbol) to indicate that is the location of the Turing Machine's tape.

- Initial setup: For the automaton, things start with the automaton being somewhere in the cloud and the state initialized to q_0
- How do we populate the stacks to represent the initial configuration of the Turing Machine?
- If we just read all the input and go back, the stack shouldn't have all the input.
- Read the input one symbol at a time and push it onto one of the stacks.

- Last character of the input sitting on the top of the stack, so flip that pile of plates.

- We need to add end of stack markers first as part of the setup.
- When we say push the input symbol by symbol, what is the input?
- There is no such thing as "the input" as far as the automaton is concerned.
- From the automaton's point of view, it needs to ready with its verdict in advance.
- Can only read input symbol by symbol.
- Can request to be provided with the next symbol of its input.

- Decide the you have grabbed enough input symbols and just run the Turing Machine on that input.
- If it so happens that we haven't read the full string, then the automaton will blow up because the accept state is unique and has NO outgoing arrows.

- At the beginning of this class, Sherstov made a very ambitious, arrogant statement
- We will understand the nature of computation.
- What do we need for general, universal computation?

Universal computation has been just around the corner!

- Started with a limited automaton, who cares?
- Pattern matching capabilities and fun to play with.
- We added a stack and all of a sudden, we have a much more realistic model that can handle natural languages and can compile computer programs
- Not universal. Things that are limited to non-PDAs and context-free grammars
- Added a second stack in this theorem, and all of a sudden, we have universal computing power.

Q. What do I need to solve any problem?
A. There is no such thing as an iPhone, no Von Neumann architecture! Cannot compute anything!

- Can you achieve general purpose computation in this model? Absolutely!

- Any organism: living or artificial, has to embody this property to be computationally universal!
    - If you are stranded on an island, you might run a billion times slower, BUT I know exactly what universal computing power is.

In 100 years, none of us will be around and this class won't be around, and people will be computing things differently.

- However, we will still be bounded by a bunch of states + 2 stacks.

Q. How do we know that there is no alternative that cannot work?
A. The most promising alternative is the quantum computer. A quantum computer cannot factor the number 15. It just doesn't have enough memory!

- People who understand quantum physics say that preserving quantum states is such a formidable challenge that general purpose quantum computing is NOT realistically feasible.
- Even if it were to be feasible, it would just speed things up a little bit.
- Even if it speeds things up exponentially, it wouldn't change what is computable and what is not computable.

People have racked their brains for decades for this question, and nobody has been able to show any model that can foil the Turing Machine.

Snap peas- highly seasonal peas in the pod

- Only available in summer
- Summer afternoon - daughter and Sherstov walk every evening
- Day after day, she follows the same kind of ritual with the peas.
- She takes two pea pods and compares them visually.
- Takes her 8 comparisons and identifies the longest of the 8 and the longer ones are more satisfying!
- When Sherstov realized what was going on, he was absolutely speechless.
- Not at all surprising that a kid age 2.5 is able to pick out the longest of 8 pea pods.

- She used an algorithm to compare the 8 pea pods.
- 2.5 year old has a solution that scales!
- Human civilization - running without any dwelling, clothes, or food supply.
- Completely vulnerable to the elements as an environment, but now we can build skyscrapers!

This idea of an algorithm and universal computation is fascinating

- Repeating the same small finite sequence of steps to achieve a goal is very fundamental.

Q. Two stacks vs single queue?

- A single queue can do anything that two stacks can!

- When you understand the two stack concept, you truly become a computer scientist!

14 Decidability
14.1 Basic notions

W 9 W Lec 11-23-16
14 Decidability
- PDA - compilers for programming languages
- A language as simple as ww cannot be recognized by a PDA
- PDA has one stack, so let's add another stack
- So far, every language we have been able to come up with has been recognizable by a Turing Machine
- So far, there is no reason to believe anything is outside of this circle

Decides a language if it recognizes it and always halts
- No such thing as rejection due to looping!

A language is decidable if some Turing Machine decides it
- Two ways of getting rejected from a job
- Explicit rejection
- Implicit rejection (when you don't hear back) - #Symantec
- **The Turing Machine will always give you an explicit answer!**
- Given an input, your computer will start to compute.
- The computer will necessarily halt and flash the word "YES" if it is in the language
- Otherwise, the computer will flash the word "NO"

- What if the Turing Machine never gets back to you?
- How do you know if you have waited long enough?
- There is no knowing that you waited long enough.
- If you had two Turing Machines: one that recognized M, another that recognized M complement, then you could recognize the language

You have to run them simultaneously because you might end up in an infinite loop

- This lecture and the lecture are Monday are THE most exciting lectures because of the level of creativity you need to bring to the solution process.
- Start small but build up very quickly to a level that really requires you to think outside of the box

Given a graph G, decide if it is connected
- This entire theory is easily generalized to practical problems
- Not phrased in the terminology of the language as a set of 0s and 1s

- Easily phrased in those terms by encoding the graph in binary
- Any real-world problem is a language is problem.
- On a given input, that can be represented in binary, you have to say "YES" or "NO"

Good opportunity to tell ourselves we can represent everything in binary

- Many formats for representing a graph in binary.
- Use binary labels for our vertices
- Consecutive integers in binary from 000 all the way to 101
- Label cannot be fractional # of bits long
- A sample format where you can feel free to use any format that makes sense.

Any mathematical object can be encoded in binary

- All binary encodings of graphs can be connected
- **On input the encoding of a graph, decide if it is connected.**

A language is a set of strings

Q. How would you define a Turing Machine that decides connectivity?
A. You have to decide whether the string represents a connected graph or a disconnected graph! Don't have to be efficient, btw!

- Computability is decision in principle and does NOT have to be efficient.
- Find a set of edges from one vertex to every other vertex.

For every two vertices, find a path that connects them.

- Search the using brute force search but there are certainly many other solutions.
- On the end of the tape, write down the first node of the first edge, and then for every edge, add nodes to that set to the end of the tape and you keep checking the entire set with every edge as two nodes.
- Do that for every edge.
- It may lure your Turing Machine into an infinite loop, and we need to make sure the format is legitimate to start with.

<u>Deciding L</u>: Not the most efficient algorithm but there is at least one algorithm that suffices to show the language L is decidable

The fact that this DFA can use any arbitrary set of 0s and 1s was a stumbling block that people couldn't comprehend at first.

- Really fun the first time you see it, but if you keep doing it, it gets old and repetitive and boring!

- No white space, line breaks, spaces, etc.
- Just for clarity!

Next two declarations tell you how you determine your alphabet.

- The next sequence of lines will be delta in lexicographic order

Self-punctuating: You can read the symbols one at a time and then at any point in time, you know the encoding is complete or you can request the next symbol.

- This is by no means the efficient format of representing DFAs, but it can be represented in a straightforward way in binary.

Q. How do we know when to stop interpreting as deltas
A. Table gives you delta in lexicographic order. Last pair of state symbol and the last pair of state and current symbol.

Exam 3 Results

- 41% A's
- 33% B's
- 1 in 4 students got a perfect score.
- Almost as high as Exam 1
- On Day 1, he will not curve against you.
- Anything above 90% is some version of an A.
- Takes Sherstov about a week to prepare an exam and the Das give feedback and prepare a new version of the exam if necessary
- Underground cycle running that the students don't know of.
- Context-Free Grammars are the least intuitive of these

For those who didn't do as well, this is the last opportunity to step it up and take advantage of the office hours.

- One fewer discussion section than usual.
- Park in the TA office and have them answer questions about hw.
- Really take advantage of office hours.

Average is about 16.8

- Nobody got 25 points, but one person got 24.75

#

You would know if the encoding is finished or you would have to encode the next symbol

- When you have two things that are self-punctuating, you can write them next to each other and blend them into an object that didn't exist.
- You can easily omit this symbol and the encoding will still make sense and you can still parse it.

DFA D example

- Run the DFA on the string w!
- Why is it okay to use three tapes?
- You can simulate any # of tapes with a single tape

- Sherstov's first programming language was BASIC
- You never number your lines consecutively because what if you need to insert a line

- Language of Turing Machines is really the language of programs

• Treat your NFA as a graph problem and see if you can reach the source vertex with a graph of this form

- We know how to convert a regular expression to an NFA, so let's do that!

Ex. 4

• This is a graph problem! You have to think back to the first exam since this is a variation of one of those problems

Ex. 5

• The TAs have to do this for HW and exams in order to award points!
• The solution is actual very short.

Jonny: Use reduction in Myhill-Nerode and break it down simpler!

Ex. 6

• Cycles may be odd or even, so we cannot really use a graph algorithm here
• Create a DFA of strings that recognizes even length D'

Ex. 7

• Language accepts if they are reversed
• What if you interest L with L^R and see if that language is the same as L

Ex. 8

• Is a language infinite?
• Check if there is a loop in relevant area of a graph!

W 10 M Lec 11-28-16

• For a language to be decidable, if the input is in the language, the TM must explicitly say "YES", and if the input is NOT in the language, the TM must explicitly say "NO".

13.4 Decidable problems about grammars

• Requires the TM to terminate and tell us if it is empty or NOT empty.
• What would be a simple solution if they ask you for a TM that asks you to recognize grammars that generate at least 1 string
• If the grammar generates a string, terminate and say "YES"
• Otherwise, all bets are off because you cannot conclude this.

- Non-deterministically apply the rules and if you generate a character, say "YES"
- **Non-determinism doesn't extend the computational power of the Turing Machine**

If you never bottom out, big deal! Maybe the grammar doesn't generate any strings and you were unfortunate in your choices of which variables you expanded in which order.

Q. In the grammar, could you look when the r.h.s. is all terminals and look for any paths to that r.h.s.
A. You need to look at some basic structural facts about those grammars

- If somebody asks you if a grammar generates infinitely many strings, that is hard!

Given a grammar, you can give an a priori upper bound of that parse tree.

- There has to be some repetition in the parse tree.

Solution to 1
- Systematically enumerate all possible paths of depth <= |V|
- If it did generate a string, it would generate it with a short parse tree

Solution to 2
- If you adapt the same technique, we look at the length of string w!
- We can also consider that the grammar would generate a relatively short parse tree if it exists!

- If you have a DFA or NFA, you could systematically try all possible executions (finitely many ways you could branch)
- Once we have a PDA, we are really in trouble because a PDA has this infinite stack and as you meander around this PDA, the stack can grow arbitrarily high.
- Infinitely many possibilities for how a PDA could accept a given string.
- Go this round-about route to see if a PDA accepts a string.

- The only way we know how to check if a PDA accepts a string is to convert a PDA to a grammar and apply it to the solution.
- Given a PDA, you would think of systematically trying out solutions but you cannot do this!

# Try out the sample final exams from previous years and you can see the level of details from these problems!

Problem 3 Solution
- There is an actual formula that you can compute.

- Look at the original grammar and see what its pumping length is.
- # of variables ^ (longest length of any r.h.s. in any rule of your ruleset)
- Obtain a grammar for the CFL which is the intersection of the language of G {L(G)} intersected with sigma^k sigma^*
- Check to see if it generates at least one string using the solution to the first problem.

Ex: Find the grammar of palindromes, intersect with the language for this, and check if it is empty?

- If you were content with just recognizing a TM, what would you need?
- See whether the DFA accepts some palindrome, and see if it does or not?

- Just enumerate the palindromes in lexicographic order and try them out one by one.
- Use brute force i.e. the empty string and see if it accepts 0 or 1. Then keep trying bigger palindrome options.
- If you do NOT find a palindrome that is accepted, you know you will never terminate, and that is a big problem!

- Even though ostensibly, this problem requires you to decide a question, you have to build up the knowledge that you gained while studying context-free grammars and synthesize stuff you learned throughout the quarter.
- A relatively efficient way of gauging your understanding of material throughout the quarter.

Ex: Use a graph search algorithm to check if any state is unreachable.

- Use a graph search algorithm like BFS or DFS.
- Does this NFA have a state that cannot be reached by any computation?
- Use a graph-search algorithm to check if there is unreachable states.
- This would NOT work with a PDA because we have an infinite stack that can go very high up.

- Convert PDA to a grammar and check if that grammar generates any strings!

Sometimes, questions are unanswerable :(

- Not a computer program or anything that can answer something.
- If someone givens you a CFG and asks you if the CFG generates all strings.

13.5 Computing a function

- A Turing Machine M computes f iff when you start M on a device and when you halt, the tape contents will be exactly f(w)

# Now is a good time to check any discrepancies in the grade book.

- With 80 students and so many scores, it is a massive amount of data.

- There are close to 1,000 entries in that table that they enter manually

Final Exam Info:
Monday 8 AM Exam

- Tuesday to Wednesday, the exam is graded.
- Thursday to Friday, leave a regrade request and check what you got.

- Make sure that all of you get a chance to raise your graded final exam and urge you to take advantage of these opportunities and send an email with the office hours for finals week.

15 Undecidability

- Languages outside our known spectra exist in large amounts
- Actual, practical problems abound that are non-decidable.

Cantor - genius from the Middle Ages

- All problems that actually matter have algorithms
- Theoretical results that show the existence of an undecidable language.

Proof: Gigantic table

3:28 PM on 11/28/16

- You will have forgotten most of what you learn from here, but you will remember this moment because it will change the way you think about computing!
- Language of all strings w s.t. w the string is rejected by w the TM
- You can take a computer program and run it on itself as input was a conceptual breakthrough!
- Revolution in how we think of computing.
- Try running Microsoft Word on Microsoft Word
- The fact that you can do it has tremendous implications for computing.
- Constructed L artificially to trick every single TM

Q. How do you prove such an arbitrary table does exist?
A. What a given TM does on any given string is fixed! It either accepts it or rejects it. Either the entry is a “YES” or it is a “NO”

- Once we fix the encoding of the Turing Machine it is completely fixed.

15.2 Halting Problem

- We don’t know what strings are in this language, but we know it exists!
- Does the program terminate in this data?
- It turns out that this eminently practical problem is NOT decidable!
- In terms of method, it is a straightforward corollary of the previous result.

Proof:

- Run M_x on x and output the opposite answer

- If x the Turing Machine halts on x the string, we are going to run it on that string.
- Whatever it outputs, we will output the opposite.
- The fundamental stumbling block to deciding L is being able to check whether a computer program halts on particular (given) data.

Questions to ask Sherstov

- Page 127 - 12.5 Last Example - ask a visual example of this
- **Page 134 - Top Example**
- Page 135 - What if we didn't have read once access to input
- Page 136 - Initial Setup
- Page 141 - What is dw?
- **Page 143 - I don't understand the parse tree proof**

W 10 W Lec 11-30-16

- 3:28 PM on November 28

The proof is the formula.

- Magical situations that happened in ancient times.
- Well-defined scenario once we agree on how to represent programs as binary strings
- Compiler
- Completely determined and fixed what any TM gives on any input.
- w either accepts w, or it doesn't!
- L is the set of all w s.t. w the string is NOT accepted by w the Turing Machine
- Suppose there is a TM that recognizes it, that TM is w!
- The TM w is going to give us the wrong answer on string w.
- If it says "YES" on w, it is wrong because w is NOT in the language by definition

- The TM w means the TM
/encoded by the string w.

Every string is uniquely defined, so there is NO conflicts in the definition.

- This phenomenon arises in many guises, but we call it that.

- Does the program terminate on the data?
- Before you launch a program on any dataset, it may be days or weeks before they compute.
- MATLAB scripts analyze huge amounts of data.
- Just running a program like this with tons of servers involved costs huge amounts of money and time.
- You cannot do it!

- For a specific cherry-picked collection of problems, we are getting back to that point because it is so unsettling.
- Computer scientists want to automate things!

Why is this undecidable?

- If you could tell on any given input whether the program terminates, you could decide this language as well.
- Check if w the program terminates on w the string using the hypothetical decider for HALT.
- Check to see if w the program halts on w the string.
- If it does halt, say "YES" w is in L
- The only obstacle to deciding this language is knowing whether a given program terminates on given data.
- Functionality of deciding is also undecidable.

HALT

- If it halts, then it ends in q_accept or q_reject
- If it doesn't halt, it goes on forever.

HALT complement

- You could decide a language by simultaneously running a recognizer for a TM and its complement

- By interchanging the roles of L and L complement, L complement is also decidable

What other problems are off-limits to computation?
A. Rice's Theorem

- We don't have general solutions for problems that matter.

15.6 Rice's theorem

- Function that returns true or false
- You can invest an hour and figure out on empty input, it gives true or false
- After some undetermined time period, you will figure out the answer.

- Does it accept every string or does it accept finitely any strings?

Someone gives you a computer program and asks:

- Does the language of the program have property P?

Rice proved that there are only two situations when you could answer this question:

0. The property is enjoyed by all programs
0. The black circle fills out the entire containing circle
0. The property is enjoyed by no programs

- What doors what this open?
  - You could decide the halting problem!
  - Get a decider for HALT

- How would you recognize a Java program with this functionality?

A. Powerful ally with a decider for P

- Does the given program have property P?
- A. Leverage that property to answer the following question
- Create a program that will be customized and tailored to T and w.
- Create it at runtime and see whether it has property P.
- Set it up s.t. T has property P if T halts on w

At your disposal is this black box for some other functionality.

- Whip up a program at runtime.
- This program has, hardcoded into it, T and w
- T and w are hardcoded constants in this program that are defined elsewhere in the outside world.
- Create the blue script in runtime with T and w hardcoded into it.

Q. What happens if T never halts on w? What is the language of this script?
A. Script does NOT terminate on any input so language is **EMPTY**

- Language of blue script is the language A, so this first line makes it so that either the blue script implements the empty language, or the blue script implements the language A
- Depending whether T halts on X
- Stare at it and try to get used to it.

- Assume you had a blackbox at your disposal and use it to build a bigger machine that can recognize HALT
- Create on the fly a program and feed it functionality.

- What if empty is here?
- Language A is orange and all we care is that one of the languages in this pair has property P and the other doesn't

- Assume this blackbox, it makes it possible to tell for any TM if T halts on w, so it makes it possible to solve the Halting problem
- Feed blue script as the input for the decider
- If T holds on w, the first line is a No-op and the real action is in line 2.
- This first line is like a switch.

Q. Does the decider give the output if it is empty or is it the blue script?
A. Report the decider's output verbatim, and whatever the decider outputs will be the answer.

Q. Out of the pairs of languages, one of them always has to be the input?
A. Correct! Empty language has special status here.

# Study exams on the webpage

- Give high-level descriptions of TM
- Give transition diagrams of a **SMALL** TM
- Be able to give algorithms for solving problems for DFAs and PDAs
- Does this PDA recognize the empty language?
- If you look at the posted exams, there is a huge variety of practice problems, so practice solving those problems and the solutions provided in those exams.
- If you can handle the previous exams, you are in very strong position.
- If someone asks you to implement some functionality, be prepared to do that.
- If someone gives you a Java program, does it accept 2016 zeros? If so, give an algorithm, or prove that it is undecidable.
- The amount of content will be similar to exams 1-3 but here, you may want to take advantage of the extra time.
- When they grade the exams, try to award as many partial credit points as possible.
- More detail is better.

#

15.7 Software Reliability

- Solve halting problem for 1 input string
- This is easier than the halting problem!
- Does the program halt on the string?
- The input string is fixed to epsilon, so we are just trying to solve the halting problem for the empty string.

- Generate a program on the fly as a script that contradicts this.
- Has its own formal argument (input) but it ignores it completely
- Creates a function on the fly (creates a text file with extension .java)
- It doesn't run it and it better not run it because who knows what the consequences of that could be
- Feeds the file textually to halt on empty.

- This script when viewed as a computer program operates the same for ALL inputs.
- Doesn't care about its input (x)

- If M halts on w, it accepts and if M does NOT halt on w, it doesn't halt.
- This script either halts on all inputs (including epsilon) or it does NOT halt on all inputs (including epsilon)

- Being able to solve the halting problem for the empty string enables you to solve the halting problem for any string w
- You CANNOT do this!

Q. How does the result of the blue script correlate to the halt on empty?
A. If M halts on w, then the script runs M on w and it says "YES"

- If on the other hand, M does NOT halt on w, M always says NO and gets trapped in this rabbit hole.
- It never halts on empty or any other string.

It is unknowable! If you knew, then you would be able to solve the halting problem.

- You get caught on this line and you never recover from it.
- You either run forever or you always terminate and output YES

You are just producing the text files and feeding it as input since you cannot run it.

Q. What is the input x used for?
A. The inputs to your function are stored when it is invoked from the command line. Any Java program that you create has its input like args[]

- Input x is just the name for the input to the script.

15.8 Homework grading

- Given the same input, do both programs generate the same output?

- Create a pair of programs on the fly.
- Created two programs on the fly at runtime with M and w hardcoded into them.
- **Can hardcode things because you create them on the fly**
- Text files with extension .java
- Feed these as input into the black box
- Textually, these inputs are pairs of computer programs that you can just feed them into the decider
- If M halts on w, then this line is a NO-OP

If M_1 and M_2 are context-free languages, you can't do anything!

- Is there a systematic way of determining if a given CFG is ambiguous?
- I wish! There is NOT a way to do this.
- Undecidable, so no such procedure or algorithm exists.
- The best we can hope for is a person looking on a case-by-case basis.
- Given two CFGs are they equivalent?
- Could you automate a human, could they automate these things?

15.8 Garbage collection (GC)

- Return this allocated space so that other programs can use it.
- The OS deciding at runtime whether any given memory object is going to be needed in the future

- This is important but off-limits!

- You can only collect garbage in a limited way via scoping
- Sequence of applications demystifies the notion of decidability or undecidability

What was this class about?
- We are going to understand what computation is and we understood that computation
- Computer architecture is temporary, but computation is forever!
- Much simpler and clearer notions than one might expect.
- Given the great progress in computer science such as AI, computer architecture, bioinformatics, etc., are there limits to what we can reach?
- Practical problems that we really want to solve are always off-limits to computing technology.
- Not just a limitation of computers we have today, but it is in the nature of things.
- What about us humans?
- When Sherstov took the class, he couldn't sleep.
- As a human being, can I solve undecidable problems?
- The answer was a resounding "YES"!
- If someone gave him a computer program and an input program, he could easily figure out if it would Halt or not, but it was bounded by some finite time.
- If it was a program with 10,000 lines of code and he dedicated his mind to it, Sherstov could do it (genius)
- There are all these theorems that claim the opposite and no device in nature can solve this.
- Sherstov is a biological computational model, so this is a statement of him as well.
- Cognitive dissonance in him and food didn't taste right.

Intimate matter for Sherstov, and it is something you have to live through.
- Philosophical question
- To claim you can solve a problem, you cannot solve every instance of the problem because life is too short.
- Life is too short, you cannot handle them all, so for Sherstov to make this astounding claim that he could solve the halting problem would have to have something to show for it.
- Leave something behind because he is NOT going to be around to solve every instance.
- The only proof is a recipe (aka algorithm) and that just doesn't exist.
- It is satisfying and comforting in closure.
- Humans are subject to the same limitations as Turing Machines
- Something that we will have to struggle with and come up with an answer.

Final Review Notes

Undecidable

- If TM is given an input, if you have to write particular symbol on tape
- If you have a useless state (never enters)
- If you have an input that is at the very end (all the way at the left and your machine moves left or all the way at the right and your machine moves right)

Decidable

- If it is possible to move and loop through