

CSE 105 Discussion – Week 7

Turing Machine Review, Computational Problems, and Decidability

Turing Machine Definition

DEFINITION 3.3

A *Turing machine* is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the *blank symbol* \sqcup ,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Given a state q and a character c under the tape head, this tells you which state q' to move to. Moreover, what character c' to write down on the tape and which direction to move the tape head.

Compared to Previous Machines...

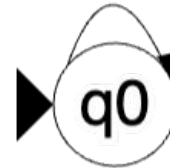
- Turing Machines process input differently (not necessarily left to right)
- Accept is accept immediately! Same with reject
- Random access to read/write memory. (PDA only has access to a stack, cannot read or write however it wants)

Because of the above, we can get stuck in a computation forever!

$\square \rightarrow \square, R$

Your computer can do this too, for example:

```
while(1);
```



Multiple Descriptions

- 7-tuple description $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$
- Implementation level description
 - Basically talking about how you modify the tape without mentioning the states
 - Can use if/else/loops
- High-level description
 - Anything as long as it is “doable” by a Turing Machine

Example

- Recall that $\{w\#w^R \mid w \text{ a string}\}$ is context free, but its cousin $\{w\#w \mid w \text{ a string}\}$ is not.
- Implementation level description of a TM that recognizes this language:
 - Start by checking if the first character is a # or not. If it is, accept iff the next one is a blank
 - Otherwise, cross off the first character with an X and move right until #. If not found, reject
 - Find the next character after # and mark it with an X iff it matches the first character
 - Go back to the start and repeat zigzagging and matching corresponding characters
 - If at any point we find a mismatch or missing/extra characters after #, reject. Accept otherwise
- High-level description
 - Accept iff the string to the left of # is the same as the string to the right

High-level Descriptions

- We see that it's really overkill for simple languages.
- Usually used so that one can call other Turing Machines as subroutine
 - Let A and B be Turing Machines
 - Let C also be a Turing Machine that on input x:
 - Run 11 on A. If accept, accept. If reject, go to second step
 - Run x on B. If accept, reject. If reject, accept.
- There is so much machinery required to set up the Turing Machine C to run A and B. But with high-level descriptions, we are not bogged down by implementation details
- Is C a decider?

Decider and Decidable Language

- Def: Deciders is a class of Turing Machine that always halt no matter the input
- A language is decidable iff there exists a decider that recognizes it
 - A language cannot be a “decider”. Similarly, a Turing machine can not be “decidable”
- Does C always halt?
- What condition do we need on A and B so that C always halts?

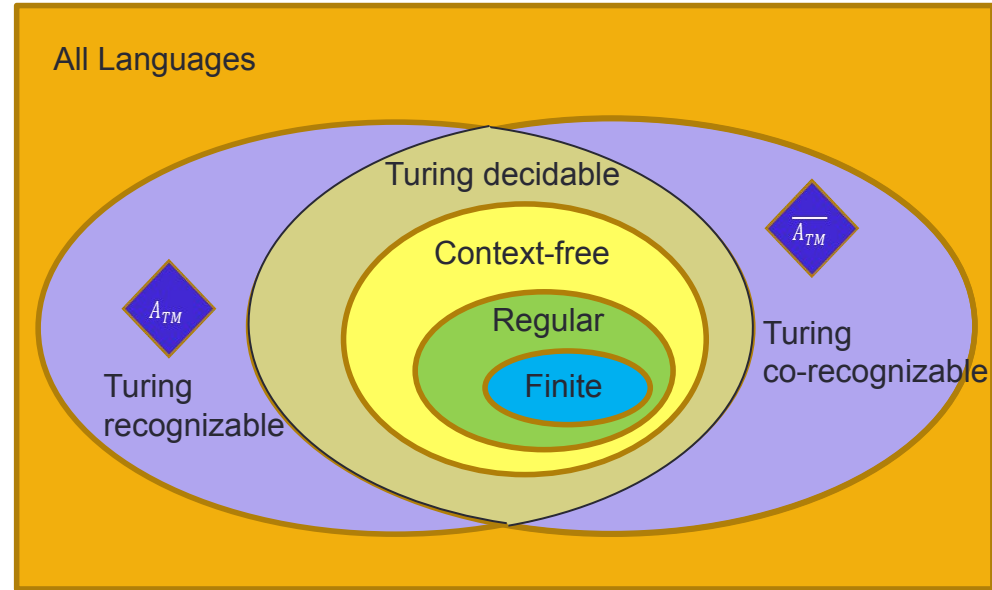
Ans1: No! C doesn't always halt because A can get stuck forever on 11. Even if A doesn't loop on 11, B can loop on x!

Ans2: We need A to halt on 11 and B to be a decider.

- Let A and B be Turing Machines
- Let C also be a Turing Machine that on input x:
 - Run 11 on A. If accept, accept. If reject, go to second step
 - Run x on B. If accept, reject. If reject, accept.

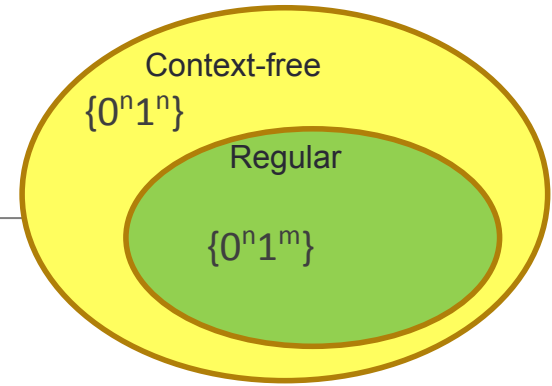
Properties of Languages

- Regular
 - Can be recognized by a DFA/NFA
 - Can be described by a Regex
- Context-Free
 - Can be recognized by a PDA
 - Can be generated by a CFG
- Recognizable
 - Can be recognized by a TM
 - Can be printed by an enumerator
- Decidable
 - Can be decided by a TM



Caveats

- Regular, decidable, etc. are properties of a language
 - Does not make sense to say a TM is decidable!
 - Does not make sense to say a string is regular!
- The diagram from last page absolutely does not mean set inclusion for a single language. We are talking about a set of languages!
 - Note in the following example $\{0^n 1^n\}$ is actually a subset of $\{0^n 1^m\} = L(0^* 1^*)$
 - How many strings is inside a language tells you nothing about the “difficulty” of deciding **which** strings are in the language.
 - However, the set of context-free languages is $\{ \{0^n 1^n\}, \{0^n 1^m\}, \dots \}$, but the set of regular languages does not contain the language $\{0^n 1^n\}$



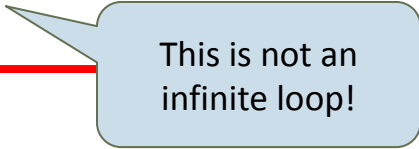
Practice

- If a language L and its complement are both recognizable, then the language is decidable. Is it true?

Yep. First, recognizable might sound fancy but it just means you can find some TM that recognizes it. So now you have M and N recognizing L and L complement respectively. What now?

We need a machine that always halt. We know if the input is in L , M halts and accepts. If input is in L complement, N halts and accepts. Can we somehow use both?

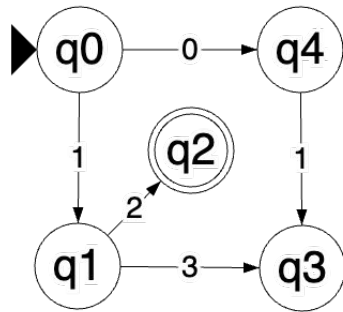
for i from 1 to infinity:
 Run input x on M for i steps and then on N for i steps
 If M accepts, accept. If N accepts, reject.



This is not an infinite loop!

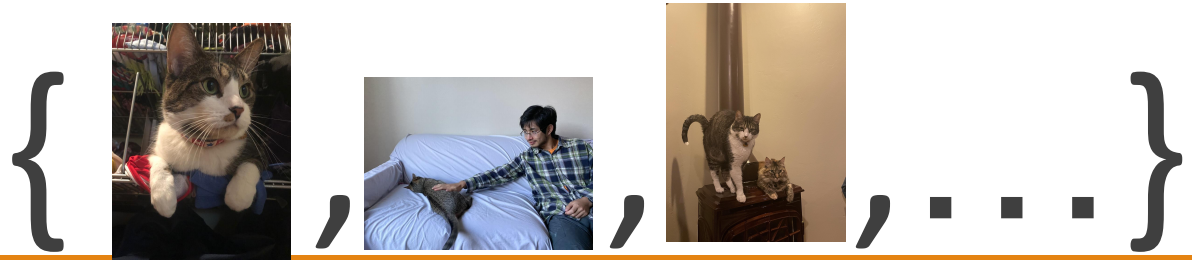
Switch of Topics: Decision Problems

- A problem with finite input and has a yes or no answer.
- Examples:
 - Is this a picture of a cat?
 - Does the graph contain a cycle?
 - Does this NFA accept this string?



Using Machines to Solve Problems

- We don't just want to solve a single instance of a problem
 - It wouldn't be useful if your computer program only accepts a fixed input
- Need a way to represent the problem
 - Collect together all inputs of a problem that has "yes" as the answer
- Definition: A decision problem is decidable if the set of inputs that has a "yes" answer can be decided by a Turing machine
 - Similar definitions apply for context-free/recognizable etc.



How Can You Feed the Input?

- In the models of computation we have discussed, machines only take strings
 - Need encoding
 - We already do this with computers: every image is represented as a sequence of 0s and 1s
 - flapjs encodes a machine into a string that happens to be a [hyperlink](#)
- In this class, we don't implement encodings but instead assume they already exist
 - Use the angle bracket to denote the thing of question has been encoded into a string
 - For example, let D be a Turing machine decider. $\langle D \rangle$ would be its encoding in string (if that's too abstract still, think of $\langle D \rangle$ as a flapjs link which is a string)

Common Decision Problems

- $A_{XX}: \{ \langle M, w \rangle \mid w \text{ is accepted by } M, \text{ which has type } XX \}$ (e.g. $A_{\text{DFA}}, A_{\text{TM}}$)
- $E_{XX}: \{ \langle M \rangle \mid M \text{ is of type } XX, \text{ and } L(M) \text{ is empty} \}$ (e.g. E_{DFA})
- $EQ_{XX}: \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are of type } XX, \text{ and } L(M_1) = L(M_2) \}$ (e.g. EQ_{DFA})

Again, a decision problem / language L is decidable if there exists some Turing Machine that halts on every input x and accepts if and only if x is in L .

Examples

Is $\langle M, 0100 \rangle \in A_{\text{DFA}}?$ YES - computation ends in q_0

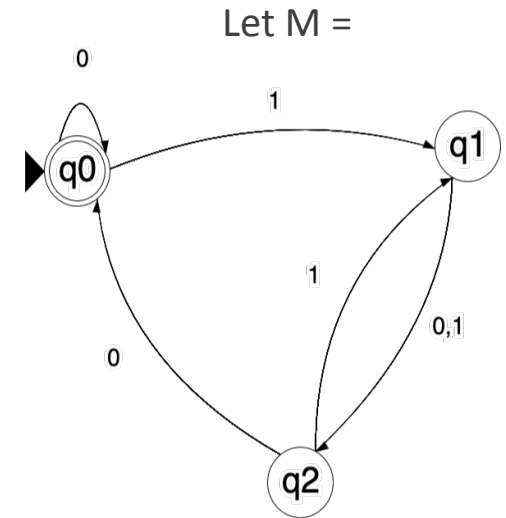
Is $\langle M, 011 \rangle \in A_{\text{DFA}}?$ NO - computation ends in q_2

Is $\langle M \rangle \in A_{\text{DFA}}?$ NO - doesn't type check

Is $\langle M, 0100 \rangle \in A_{\text{REX}}?$ NO - doesn't type check

Is $\langle M \rangle \in E_{\text{DFA}}?$ NO - accepts 0, 00, etc.

Is $\langle M, M \rangle \in EQ_{\text{DFA}}?$ YES - $L(M) = L(M)$



Practice

- We know A_{DFA} is decidable. What about its complement?
 - What even is in the complement of A_{DFA} ?
 - $\{\langle D, w \rangle \mid \text{the DFA } D \text{ does not accept } w\}$ is certainly a subset, but you also have other garbage (things not even of the right form, say $\langle D \rangle$, $\$ \# @ 5^{**\%}$)
 - Let T be a decider for A_{DFA} , all you need to do is to flip the accept/reject of T to find a decider for the complement of A_{DFA} . To put it another way, build T' such that on input x , run T on x . Do the opposite thing T does.

More Practice

- Consider the complement of EQ_{DFA}
 - What is in it?
 - $\langle A, B \rangle$ such that $L(A) \neq L(B)$ and a bunch of garbage
- Is it recognizable?
 - If the input is not of the form $\langle A, B \rangle$ where A, B are both DFAs, gladly accept. Otherwise, one can brute force enumerate all strings and check if A and B have the same behavior on each string. Once the behaviors differ, accept.
- Can you use the same strategy to design a recognizer for EQ_{DFA} ?
 - Nope! If A and B have the same language, we are now going to loop forever!
 - However, we can find A' recognizing the complement of $L(A)$ and B' recognizing the complement of $L(B)$. What's the next step?

Continued

- Apply the intersection construction on A and B'.
 - The language is empty iff $L(A)$ is a subset of $L(B)$
- Apply the intersection construction on B and A'.
 - The language is empty iff $L(B)$ is a subset of $L(A)$
- We know how to check if the language of a DFA is empty!
- Putting everything together:
 - We construct a decider for EQ_{DFA}
 - On input, type check and reject strings not of type $\langle A, B \rangle$
 - Construct A' to recognize the complement of $L(A)$
 - Construct B' to recognize the complement of $L(B)$
 - Apply intersection construction to A and B' to yield C
 - Apply intersection construction to B and A' to yield D
 - Check if C and D have empty language. If so accept. Otherwise, reject.

Undecidable problems

We know they exist. Why?

Because the set of decidable languages is countably infinite and the set of *all* languages is uncountably infinite.

It's useful to find examples of these undecidable languages because - (as you'll see later) - they make it easy to reason about the decidability of other languages that we might not know about

One such example is $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing machine that accepts string } w \}$