D5: Turing Machines

Sec 101: MW 3:00-4:00pm DOW 1018
IA: Eric Khiu

Computability Recap

- We are interested in "what problems can / can't a computer compute"
- First, we structured what we mean by "problem" by introducing formal languages
- Next, we started to tackle what "computer" means
  - We started by looking at DFAs as computational devices
  - It turns out that DFAs are a little too limited to be a general representation of a computer
  - Now we introduce Turing Machines

Agenda

- Turing Machines
- Decidability
- Counting and Diagonalization

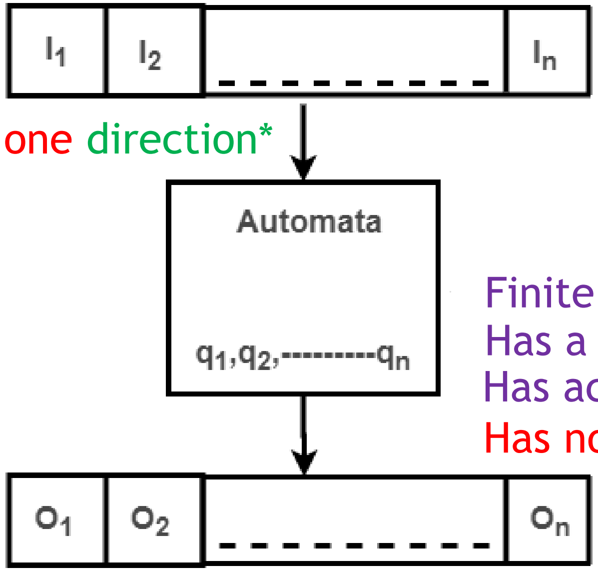
Turing Machines

Notes

Finite Automata vs Turing Machines

Finite Automata

Finite tape with finite input alphabet



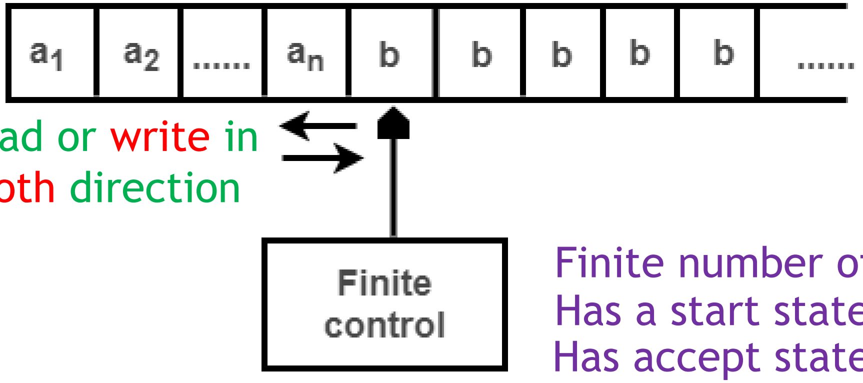
Read in one direction*

Finite number of states
Has a start state
Has accept states
Has no reject states

Halts after reading all the inputs

Turing Machine

Infinite tape with finite input alphabet and special symbols



Read or write in both direction

Finite number of states
Has a start state
Has accept states
Has reject states

Halts after reaching an accept/ reject state

Otherwise, we say it *loops indefinitely*

*The head of a *two-way automata* can move in both directions, which is out of the scope of this class

Definition and Representation

- We define a Turing machine as the 7-tuple $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$
  - Q is a finite set of ***states***
  - $q_0 \in Q$ is the ***initial state***
  - $F = \{q_{\text{accept}}, q_{\text{reject}}\} \subseteq Q$ are the ***final*** (***accept/reject***) states
  - Σ is the ***input alphabet***
  - $\Gamma \supseteq \Sigma \cup \{\perp\}$ is the ***tape alphabet*** ($\perp \notin \Sigma$ is the ***blank symbol***)
  - $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the ***transition function***

Warning: The input string cannot contain the blank symbol $\perp$ and any other symbols in $\Gamma \setminus \Sigma$!

- Turing machines can be represented with state diagrams or pseudocode
  - Turing machines are computationally equivalent to many programming languages
  - It then makes sense to use pseudocode to specify a Turing machine

TL; DPA

- We introduced the Turing machines and compared it against finite automata.
- We learned how to define a TM using the seven 7-tuple $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$.
- We established that we represent a TM using a state-transition diagram or pseudocode.
- Useful tool: https://turingmachine.io/

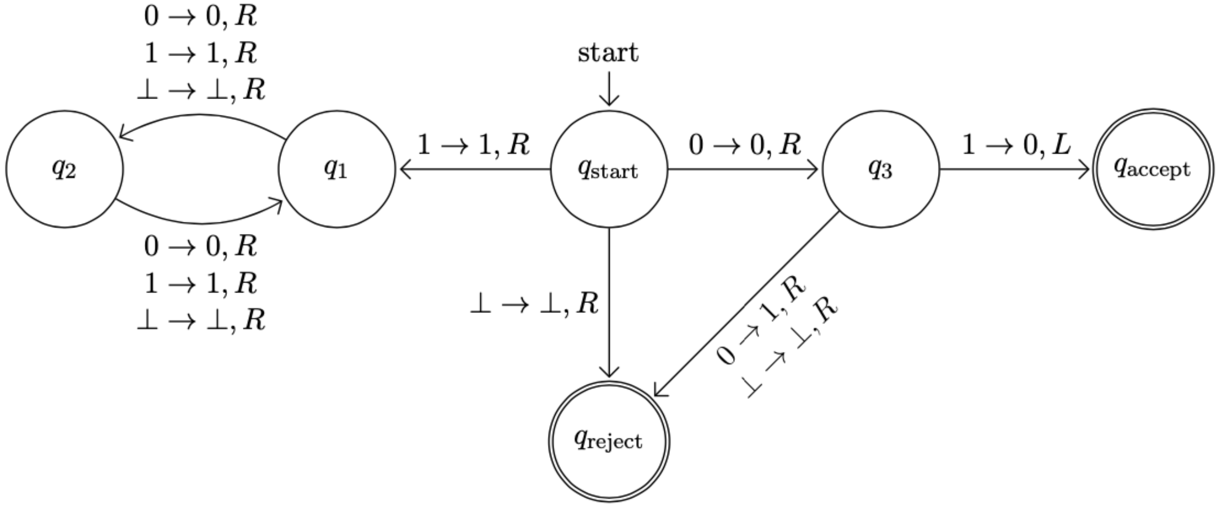
Decidability

Notes

Decidability and Turing Machines

- For a language A, we say Turing machine M decides A if:
  - For all $x \in A$, M accepts x
  - For all $x \notin A$, M rejects x
  - And M halts on all input
- Language A is decidable if there exists a TM that decides A
- We call this TM a decider of A

TM State Diagram Practice



- Does this TM accept/ reject/ loops on the following input strings?
  - ε
  - 01
  - 110
- What language over $\Sigma = \{0,1\}$ does this TM decides, if any?
  - None. Observe that if the input strings start with 1, the TM will always loop. In other words, it fails to halt on input of form 1(0|1)*

Proving Decidability

- We have established that a language L is decidable iff there exists some TM that decides L, so proving decidability = construct a TM
- Reminder 1: When we say "give an algorithm", you need to prove the correctness, this applies to TM algorithms too
- Reminder 2: To prove that a TM M decides L, we need to prove
  - For all $x \in L$, M accepts x
  - For all $x \notin L$, M rejects x
  - M halts on all input
- **Discuss:** Suppose we know some decider exists for some language, can we use it in the decider we want to construct?
  - Yes! Think of it like a *global helper function* that everyone has access to

Decidability Proof Using Known Deciders

- Suppose both S and T are both decidable languages. Prove that $L = S \setminus T$ is decidable
- Since we know that S and T are decidable, we know there exists some TMs, say D_S and D_T that decide S and T respectively.
- We can call those deciders in the TM (decider), D_L we want to build!

Correctness Analysis Draft

- Before we start writing the algorithm, let's start drafting the correctness analysis first (you'll find it useful later)
  - $x \in S \setminus T \Rightarrow \cdots \Rightarrow D_L$ accepts x $\leftarrow$ We want this to happen

Correctness Analysis Draft

- Before we start writing the algorithm, let's start drafting the correctness analysis first (you'll find it useful later)
  - $x \in S \setminus T \Rightarrow x \in S \land x \notin T \Rightarrow \cdots \Rightarrow D_L$ accepts x

Correctness Analysis Draft

- Before we start writing the algorithm, let's start drafting the correctness analysis first (you'll find it useful later)
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts x **and** D_T rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
- Otherwise, if
  - $x \notin S \setminus T \Rightarrow \cdots \Rightarrow D_L$ rejects x $\leftarrow$ We want this to happen

Correctness Analysis Draft

- Before we start writing the algorithm, let's start drafting the correctness analysis first (you'll find it useful later)
  - $x \in S \setminus T \Rightarrow x \in S \land x \notin T \Rightarrow D_S$ accepts x **and** D_T rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
- Otherwise, if
  - $x \notin S \setminus T \Rightarrow x \notin S \lor x \in T \Rightarrow \cdots \Rightarrow D_L$ rejects x

Correctness Analysis Draft

- Before we start writing the algorithm, let's start drafting the correctness analysis first (you'll find it useful later)
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts x **and** D_T rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
- Otherwise, if
  - $x \notin S \setminus T \Rightarrow x \notin S \vee x \in T \Rightarrow D_S$ rejects x or D_T accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x
- We want these two to be the only cases to ensure D_L halts on all input

TM Algorithm

- Construct D_L to make this happens:
  - $x \in S \setminus T \Rightarrow x \in S \land x \notin T \Rightarrow D_S$ accepts x **and** D_T rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \lor x \in T \Rightarrow D_S$ rejects x **or** D_T accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts **and** $D_T(x)$ rejects **then**

Accept

Reject"

TM Correctness Proof

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts **and** $D_T(x)$ rejects **then**

Accept

Reject

- We just need to complete our proof draft now!
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts $x \wedge D_T$ rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \vee x \in T \Rightarrow D_S$ rejects $x \vee D_T$ accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x

TM Correctness Proof

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts and $D_T(x)$ rejects then

Accept

Reject

Now explain what happen here

- We just need to complete our proof draft now!
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts $x \wedge D_T$ rejects $x \Rightarrow \ldots \Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \vee x \in T \Rightarrow D_S$ rejects $x \vee D_T$ accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x

TM Correctness Proof

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts **and** $D_T(x)$ rejects **then**

Accept

Reject

- We just need to complete our proof draft now!
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts $x \wedge D_T$ rejects $x \Rightarrow x$ satisfies both conditions to enter the if block, causing D_L to accept $\Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \vee x \in T \Rightarrow D_S$ rejects $x \vee D_T$ accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x

TM Correctness Proof

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts **and** $D_T(x)$ rejects **then**

Accept

Reject

- We just need to complete our proof draft now!
  - $x \in S \setminus T \Rightarrow x \in S \land x \notin T \Rightarrow D_S$ accepts $x \land D_T$ rejects $x \Rightarrow x$ satisfies both conditions to enter the if block, causing D_L to accept $\Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \lor x \in T \Rightarrow D_S$ rejects $x \lor D_T$ accepts $x \Rightarrow \cdots \Rightarrow D_L$ rejects x

Now explain what happen here

TM Correctness Proof

D_L = "On input x:

Run D_S and D_T on x

If $D_S(x)$ accepts **and** $D_T(x)$ rejects **then**

Accept

Reject

- We just need to complete our proof draft now!
  - $x \in S \setminus T \Rightarrow x \in S \wedge x \notin T \Rightarrow D_S$ accepts $x \wedge D_T$ rejects $x \Rightarrow$ x satisfies both conditions to enter the if block, causing D_L to accept $\Rightarrow D_L$ accepts x
  - $x \notin S \setminus T \Rightarrow x \notin S \vee x \in T \Rightarrow D_S$ rejects $x \Rightarrow$ x satisfies neither conditions to enter the if block, causing D_L to reject $\Rightarrow D_L$ rejects x
  - Additionally, D_L halts on all inputs because if it doesn't enter the if-block, it rejects

Decidability Proof Exercise

- Show that for any decidable language L, $L \cup \{\varepsilon\}$ is also decidable.
- Let D_L be the decider for L. We want a decider D for $L \cup \{\varepsilon\}$ with the following behavior
  - $x \in L \cup \{\varepsilon\} \Rightarrow x \in L \vee x = \varepsilon \Rightarrow \cdots \Rightarrow D(x)$ accepts
  - $x \notin L \cup \{\varepsilon\} \Rightarrow x \notin L \wedge x \neq \varepsilon \Rightarrow \cdots \Rightarrow D(x)$ rejects

Decidability Proof Exercise

Desired Behavior
- $x \in L \cup \{\varepsilon\} \Rightarrow x \in L \vee x = \varepsilon \Rightarrow \cdots \Rightarrow D(x)$ accepts
- $x \notin L \cup \{\varepsilon\} \Rightarrow x \notin L \wedge x \neq \varepsilon \Rightarrow \cdots \Rightarrow D(x)$ rejects

1. D = " On input x:
2. **if** $x = \varepsilon$ **then accept**
3. Run D_L on x
4. **if** $D_L(x)$ accepts **then accept**
5. **else reject**"

Correctness proof

- $x \in L \cup \{\varepsilon\} \Rightarrow x \in L \vee x = \varepsilon \Rightarrow D_L(x)$ accepts or D accepts on line 2 $\Rightarrow D(x)$ accepts
- $x \notin L \cup \{\varepsilon\} \Rightarrow x \notin L \wedge x \neq \varepsilon \Rightarrow x$ satisfies neither condition to enter the if-block on line 2 or line 4 $\Rightarrow$ Enter line 5 $\Rightarrow D(x)$ rejects

Decidability Concept Check 1

T/F: Given a TM M, there can be more than one distinct language L decided by M.

- False, a TM can decide either zero or one languages
- Deciders are required to halt on all inputs, so any TM that does not halt on some input is not a decider $\Rightarrow$ Decide zero language
- Now, consider TMs that are decider. The language of a decider is the set of all (finite-length) string that the machine accepts.
  - Suppose for contradiction that M decides L_1 and L_2 where $L_1 \neq L_2$
  - WLOG, $\exists x \in L_1 \setminus L_2$, i.e., $x \in L_1 \cap \overline{L_2}$
  - Since $x \in L_1$, M must accept x
  - But since $x \notin L_2$, M must reject x
  - Contradiction!

Decidability Concept Check 2

T/F: Given a decidable language L, there can be more than one distinct TM M that decides L.

- True. Consider an arbitrary decidable language L and a TM that decides it M
- Construct a different TM M' that begins by transitioning one cell right, then one cell left, not writing either time, then has an identical transition function to M
- Since M' is defined differently, $M' \neq M$
- However, M' and M both decide L
- In fact, there are infinite TM for any decidable language

Recognizability

- For a language A, we say Turing machine M recognizes A if:
  - For all $x \in A,\ M$ accepts x
  - For all $x \notin A,\ M$ does not accept x (this could mean reject or loop!)

- For DFAs, deciding and recognizing are the same
  - The TM ceases execution when it reaches accept or reject rather than the end of the input string, which leads to this distinction

TL; DPA

- We discussed the notion of decidable language- a language that is decidable by a Turing machine.
- To prove that a TM decides a language L, we show that it
  - Accepts all $x \in L$
  - Rejects all $x \notin L$
  - Halts on all inputs

Counting and Diagonalization

Notes

203 Recap: (Un)countable Infinity

- **Definition:** An infinite set X is countably infinite if you can map each $x \in X$ to a unique natural number (enumerating)
  - More formally, there is a function f such that $f : X \to \mathbb{N}$ is one-to-one (i.e. f is an *injective* function)

- If we cannot write such a function, then the set is uncountably infinite and "strictly larger than" the set of natural numbers

Definition: An infinite set X is countably infinite if you can map each $x \in X$ to a unique natural number (enumerating)

Proving Uncountably Infinite

- We use Cantor's diagonalization argument to prove that a set is uncountably infinite
- **Ex:** Prove that the set of infinite-length binary sequence is uncountably infinite
- Suppose, for the sake of contradiction, that the set of infinite-length binary sequence $S = \{s_1, s_2, \ldots\}$ is countably infinite, so we can list/ enumerate *every* sequence in an infinite table

Sequence	1st bit	2nd bit	3rd bit	4th bit	5th bit	…
s_1	0	1	1	0	0	…
s_2	0	0	0	0	0	…
s_3	1	0	1	0	1	…
s_4	1	1	0	1	0	…
$\vdots$						

Definition: An infinite set X is countably infinite if you can map each $x \in X$ to a unique natural number (enumerating)

Proving Uncountably Infinite

- Now, construct a sequence d as follows: 1st bit of s is opposite of 1st bit of sequence 1, 2nd bit of s is opposite of 2nd bit of sequence 2, … i^{th} bit of s is opposite of i^{th} bit of sequence i

Sequence	1st bit	2nd bit	3rd bit	4th bit	5th bit	…
s_1	0	1	1	0	0	…
s_2	0	0	0	0	0	…
s_3	1	0	1	0	1	…
s_4	1	1	0	1	0	…
⋮						

$d = 1,1,0,0, \dots$

Poll: The existence of d contradict the assumption that…
A. d is an *infinite-length* binary sequence
B. $S = \{s_1, s_2, \dots\}$ is a set of *all* infinite-length binary sequence
C. $S = \{s_1, s_2, \dots\}$ is countably infinite

Since we arrive at a contradiction, S must be uncountably infinite

Diagonalization Practice

- Let x, y be binary strings of the same length n over $\Sigma = \{0,1\}$. The *Hamming distance* between x and y, written $d_H(x, y)$ is the number of position $i \in \{1,2,\dots,n\}$ for which $x_i \neq y_i$. For example, $d_H(11100, 10101) = 2$ because the two strings only different in the second and fifth characters.
- Consider an infinite list of infinite binary sequences

$$\begin{aligned} s_1 &= b_{11}b_{12}b_{13}\dots \\ s_2 &= b_{21}b_{22}b_{23}\dots \\ s_3 &= b_{31}b_{32}b_{33}\dots \\ &\vdots \end{aligned}$$

- where each $b_{ij} \in \{0,1\}$. Cantor's diagonalization argument shows that the sequence $\overline{b_{11}}\,\overline{b_{22}}\,\overline{b_{33}}$ has Hamming distance at least one from every sequence in the list, where $\overline{b}$ is the complement of b.
- Construct a binary sequence that have *infinite* Hamming distance from every sequence in the list, i.e., it differ from each sequence in an infinite number of positions
  - Hint: There are infinitely many prime numbers; if p and q are distinct primes, then $p^n \neq q^m$ for all pairs of $n, m > 0$.

Diagonalization Practice

Hints:
1. There are infinitely many prime numbers
2. Let p, q be primes and $n, m > 0$. If $p \neq q$, then $p^n \neq q^m$ for all pairs of n, m.

- **Key:** Flip different bits from different sequences, but infinitely many from each
- Let p_k be the k^{th} prime number. Flip all $(p_k)^1, (p_k)^2, \ldots$ bits from the k^{th} sequence
- For example, the first prime number is 2 so we flip the 2nd, 4th, 8th, … bits in the first sequence. Since s_1 is infinite-length, $d_H(s, s_1) = \infty$.

	1st bit	2nd bit	3rd bit	4th bit	5th bit	6th bit	7th bit	8th bit	9th bit	…
s_1	0	1	1	0	0	1	0	1	1	…
s_2	0	0	0	0	0	1	1	1	1	…
$\vdots$										…
s		0		1				0		…

Diagonalization Practice

Hints:
1. There are infinitely many prime numbers
2. Let p, q be primes and $n, m > 0$. If $p \neq q$, then $p^n \neq q^m$ for all pairs of n, m.

- **Key:** Flip different bits from different sequences, but infinitely many from each
- Let p_k be the k^{th} prime number. Flip all $(p_k)^1, (p_k)^2, \ldots$ bits from the k^{th} sequence
- For example, the first prime number is 2 so we flip the 2nd, 4th, 8th, … bits in the first sequence. Since s_1 is infinite-length, $d_H(s, s_1) = \infty$.
- The second prime is 3 so we flip the 3rd, 9th , 27th, … bits in the second sequence. Again, we have $d_H(s, s_2) = \infty$

	1st bit	2nd bit	3rd bit	4th bit	5th bit	6th bit	7th bit	8th bit	9th bit	…
s_1	0	1	1	0	0	1	0	1	1	…
s_2	0	0	0	0	0	1	1	1	1	…
$\vdots$										…
s		0	1	1				0	0	…

Hints:
1. There are infinitely many prime numbers
2. Let p, q be primes and $n, m > 0$. If $p \neq q$, then $p^n \neq q^m$ for all pairs of n, m.

Diagonalization Practice

- **Key:** Flip different bits from different sequences, but infinitely many from each
- Let p_k be the k^{th} prime number. Flip all $(p_k)^1, (p_k)^2, \ldots$ bits from the k^{th} sequence
- For example, the first prime number is 2 so we flip the 2nd, 4th, 8th, … bits in the first sequence. Since s_1 is infinite-length, $d_H(s, s_1) = \infty$.
- The second prime is 3 so we flip the 3rd, 9th , 27th, … bits in the second sequence. Again, we have $d_H(s, s_2) = \infty$
- By hint 1, we can keep this going because we have infinite primes
- By hint 2, since $p_i \neq p_j \Rightarrow (p_i)^n \neq \left(p_j\right)^m$ for all pairs of n, m, there is no collisions in the index of bits flipped
- Therefore, $d_H(s, s_k) = \infty$ for all $k = 1, 2, \ldots$, as desired.

Definition: An infinite set X is countably infinite if you can map each $x \in X$ to a unique natural number (enumerating)

Proving Countably Infinite 1

- To prove that a set is countably infinite, we can demonstrate a way to enumerate the elements
- **Ex:** Show that the set consisting of all the (finite-length) ASCII strings is countable.
  - Hint: There are 128 ASCII characters which is a **finite alphabet**, thus the number of strings of length k is 128^k
- **Enumerate:** Shortlex – list the strings by length, then lexicographical order
- Create a list of all strings of each length, then concatenate them together

Length	List	Number of elements	Index of last element
0	[ε]	$128^0 = 1$	1
1	['a', 'b', ...]	$128^1 = 128$	129
⋮	⋮	⋮	⋮
k	['aa...a', 'ab...a', ...]	128^k	$\sum_{i=0}^{k} 128^k$ ← This is finite!

Therefore, we can map finite-length ASCII strings to natural numbers ⇒ Countably infinite

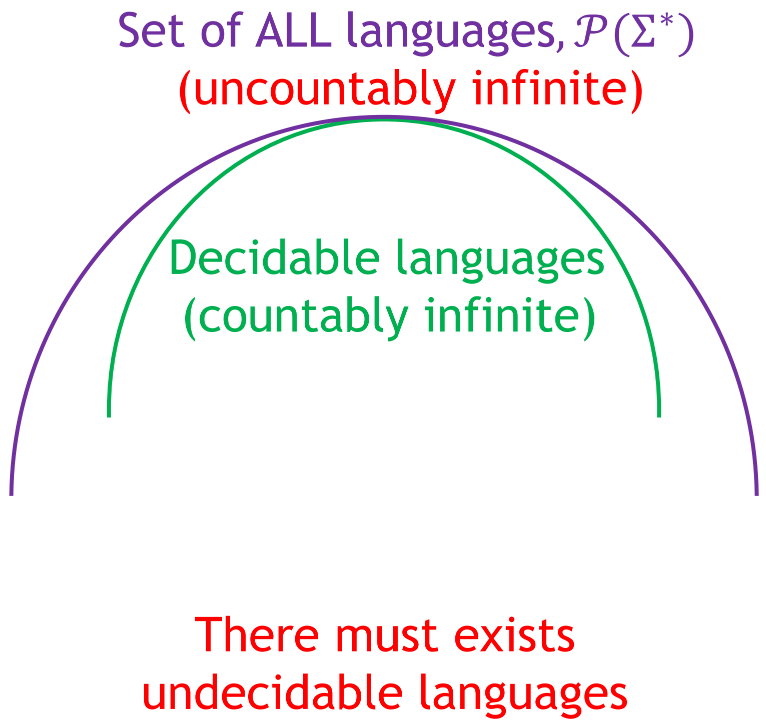
Definition: An infinite set X is countably infinite if you can map each $x \in X$ to a unique natural number (enumerating)

Proving Countably Infinite 2

- Now, prove that the set of all decidable languages over a given alphabet Σ is countable
- Hint: A TM can be represented by a finite-length ASCII string
- Previous: Proven set of all finite-length ASCII strings is countably infinite $\Rightarrow$ set of all TM is countably infinite $\Rightarrow$ we can assign TM to natural numbers
- Know: Each decidable language has at least one unique TM that decides it
  - Previous: No two TMs decide the same language
  - In fact, we have infinitely many TM for one language, but we just need one here
- Map each decidable language *arbitrarily* to one TM that decides it
- Thus, we can map decidable languages through TM to natural numbers $\Rightarrow$ countably infinite

Existence of Undecidable Languages

- We've shown in lecture the existence of undecidable languages, we now present a counting argument
- Previous: the set of decidable languages is countably infinite
- The set of strings a TM decides is $L(M) \subseteq \Sigma^*$, so the set of all languages is $\mathcal{P}(\Sigma^*)$
  - Power set of countably infinite set is uncountably infinite (will prove this in HW3)
  - The set of all languages is uncountably infinite
- Therefore, there must exists some undecidable languages



Back Matter

Diagonalization for Undecidability

- In lecture, we proved the existence of undecidable languages using the diagonalization method.
- The idea of diagonalization is not limited to reason about the countability of an infinite sets, we could also use it to prove undecidability a language.
- For example, we can use it to prove that the language $L_{ACC} = \{(\langle M \rangle, x): M \text{ is a TM that halts on } x.\}$ is undecidable.

Diagonalization for Undecidability

- Suppose for contradiction that L_{ACC} is decidable with a decider H.
- In the following table, we list all Turing machines (which we know is countable) down the rows $M_1, M_2, \ldots$ and all their description across the columns $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$.

$$\begin{array}{c|ccccc}
 & \langle M_1 \rangle & \langle M_2 \rangle & \langle M_3 \rangle & \langle M_4 \rangle & \cdots \\
\hline
M_1 & accept & & accept & & \\
M_2 & accept & accept & accept & accept & \\
M_3 & & & & & \cdots \\
M_4 & accept & accept & & & \\
\vdots & & & \vdots & &
\end{array}$$

- The entries tells whether the machine in a given row accepts the input in a given column
  - The entry is *accept* if the machine accepts the input
  - The entry is blank if it rejects or loops on that input

Diagonalization for Undecidability

- Now, we construct a similar table to capture the behavior of H (decider for L_{ACC}) on each pair of input $(M_i, \langle M_j \rangle)$, i.e.,
  - The entry is *accept* if M_i accepts $\langle M_j \rangle$
  - The entry is *reject* if M_i rejects or loops on $\langle M_j \rangle$
- For example,

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$		$\vdots$			

Table 1: (i, j) entry = $M_i(\langle M_j \rangle)$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Table 2: (i, j) entry = $H(M_i, \langle M_j \rangle)$

Diagonalization for Undecidability

- Now, construct a "diagonal" Turing machine D as follows:
  - D calls H as a subroutine to determine what M does when the input to M is its own description $\langle M \rangle$
  - Once D has this information, it does the opposite
- Essentially, we are constructing a row for TM D in Table 2 by flipping the diagonal, i.e., $(D, j) = \neg H\left(M_j, \langle M_j \rangle\right)$
- The contradiction occurs where the point at the point of the question mark where the entry must be the opposite of itself
- In essence, H cannot be a decider of L_{ACC} because it fails to predict $D(\langle D \rangle)$ (i.e., the '?' on Table 2 is undefined)

$D =$ "on input $(\langle M \rangle)$:

Run H on input $(\langle M, \langle M \rangle \rangle)$
if H accepts **then**
    reject
else
    accept"

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$		$\vdots$					$\ddots$

Table 2: (i, j) entry = $H\left(M_i, \langle M_j \rangle\right)$

Diagonalization for Undecidability

- **Warning:** Don't get confused by the notion of running a machine on its own description!
  - This is similar to running a program with itself as input, something that does occasionally occur in practice: e.g., a compiler
- In our case, we have

$$D(\langle D\rangle) = \begin{cases} accept & \text{if } D \text{ does not accept } \langle D\rangle \Leftrightarrow H(D,\langle D\rangle) \text{ rejects} \\ reject & \text{if } D \text{ accepts } \langle D\rangle \Leftrightarrow H(D,\langle D\rangle) \text{ accepts} \end{cases}$$

- Since D's design ensures that $H's$ prediction is always wrong, H cannot exists as a decider for L_{ACC}
- Therefore, there do not exists a decider for L_{ACC} and hence it's undecidable

Equivalence of 2-Tape Machines

Notes

2-Tape Turing Machines

- A two tape Turing Machine is very similar to a one tape, except that it has two input tapes with one head over each tape
- This means for each step of execution, the transition function looks at both heads, writes a character to each tape, and moves each head left or right
- (Aside) When a computation device is equivalent to the classic Turing Machine, we call it Turing Complete
  - Some examples: C++, Python, Java, Conway's Game of Life
  - Staff favorites: Minecraft, origami, PowerPoint, Magic: the Gathering

Q: Is a 2-Tape TM Turing complete, i.e., equivalent to a 1-Tape TM?

Proof of Equivalence

- Equivalence proofs need to show two directions:
  - Machine one can simulate every function of machine two
  - Machine two can simulate every function of machine one
- Direction 1: 2-tape machines can simulate one-tape machines
  - Ignore the second tape

Simulating a 2-Tape Machine on a 1-Tape

- Let $\mathcal{M}$ be an arbitrary 2-tape Turing Machine and
  Let $\mathcal{T}$ be an arbitrary 1-tape Turing Machine

Schematic of the tape for T:

$$\# \underbrace{\cdots \quad \bullet \quad \cdots}_{\text{Contents of Tape 1}} \# \underbrace{\cdots \quad \bullet \quad \cdots}_{\text{Contents of Tape 2}} \#$$

- Pseudocode:
  1. Put the tape in the correct format $\# \quad \dot{w}_1, \cdots, w_n \quad \# \quad \dot{\perp} \quad \#$
  2. Have $\mathcal{T}$ scan from the first # to the third # to find the values under the heads
  3. Make a second pass, updating the heads according to $\mathcal{M}$'s transition function
  4. If $\mathcal{T}$ tries to overwrite the middle #, shift the entire second tape down one cell