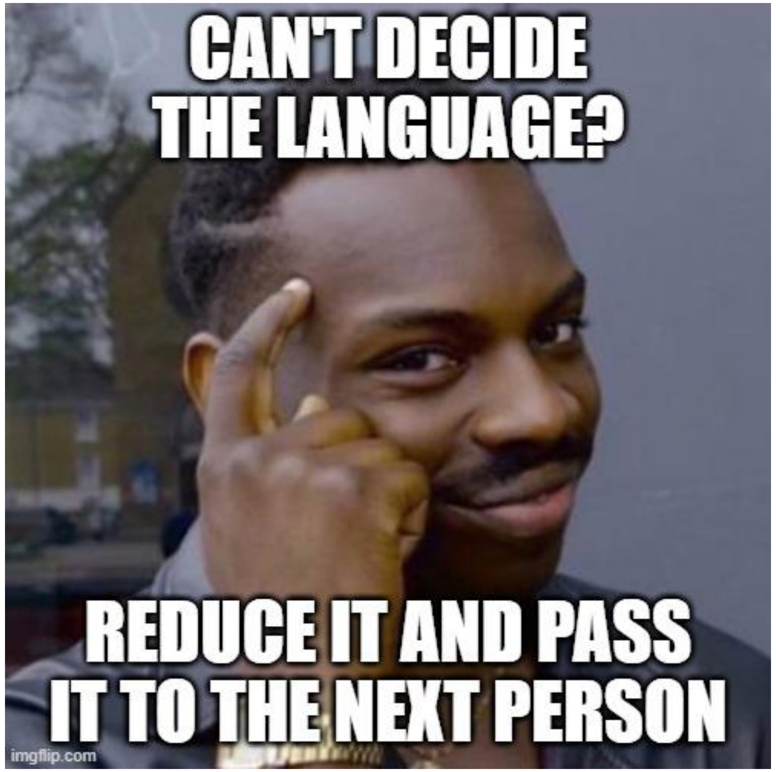
D6: Turing Reductions



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

Waquackquack is hacking the BBB Blinken Bulbs!

- Waquackquack is in Florida now, attempting to hack the BBB Blinken Bulbs. Unsure if his code worked, he consults the mysterious Oracle, which accepts a query (i, c) if bulb i is displaying color c and rejects otherwise.
- Suppose he uses the following TM to query the oracle:

D = "On input w:

    for $i = 1$ to 100 **do**

        if i is odd **then**

            if $\mathcal{O}(i, \mathrm{blue})$ rejects **then reject**

        else

            if $\mathcal{O}(i, \mathrm{yellow})$ rejects **then reject**

accept"

Waquackquack

Q: What pattern is Waquackquack aiming to achieve with the light bulbs?

Agenda

- Turing Reductions
- Proving Undecidability
- More Turing Reductions

Recap: Decidable Language

- A language L is decidable iff there exists a TM that
  - accepts all $x \in L$
  - rejects all $x \notin L$
  - halts on all inputs
- One way to prove that a language is decidable is by describing a TM that decides it (i.e., a decider)
- **Note 1:** You are allowed to hardcode constant values in your machine (seen in HW3)
- **Note 2:** If some deciders are known to exists, you can use them in your machine (e.g., D_S and D_T from last week for $S \setminus T$)

Today: Proving *Un*decidability

- We know that the following languages are undecidable:
  - $L_{BARBER} = \{\langle M\rangle : M \text{ is a TM and } M \text{ does not accept } \langle M\rangle\}$
  - $L_{HALT} = \{(\langle M\rangle, x) : M \text{ is a TM and } M \text{ halts on } x\}$
  - $L_{ACC} = \{(\langle M\rangle, x) : M \text{ is a TM and } M \text{ accepts } x\}$
- Now, say we want to prove that L is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L is decidable
  - …[something happen]…
  - Therefore, [insert undecidable language] is now decidable. Contradiction.

Turing Reductions

Notes

Turing Reducible

- A language A is *Turing reducible* to language B, written as

$$A \leq_T B$$

which means

- If, I have a TM that decides B, say $\mathcal{O}_B$ (black box/ oracle)
- Then, I can decide A

Warning: This tells me the *relationship* between languages A and B, I don't know anything (most importantly, decidability) of A and B *individually*

Turing Reductions Example

- Let A and B be defined as follows:

$$A = \{a^n : n \geq 0\} \quad B = \{b^n : n \geq 0\}$$

- **By using blackbox decider of $\boldsymbol{B}$**, prove that $A \leq_T B$
- Want to show: If I have an oracle $\mathcal{O}_B$ for B, then I can decide A.
- To do so, we build a decider for A, say D_A that uses $\mathcal{O}_B$.

- **Step 1: Identify the inputs of $\boldsymbol{D_A}$ and $\mathcal{O}_B$**
  - We will pass in a string, say x, into D_A and check if $x \in L_A$
  - We will pass in a string, say x', into $\mathcal{O}_B$ and check if $x' \in L_B$
  - **Note:** x' *may* be the same as x, but we don't know yet so assume they're different for now

Desired Behavior of D_A

- **Step 2: Draft Desired Behavior of D_A** (can be used as correctness proof later)
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow D_A(x)$ rejects
- Now, let's work backwards from the other direction.
- Using the blackbox decider essentially means using the output of that decider
- Which means we have two choices:
  - Do the same as the blackbox decider ("return same")
  - Do the opposite as the blackbox decider ("return opposite")
- Often, one choice is more intuitive than another, so just pick one and try!

Desired Behavior of D_A

- **Step 2: Draft Desired Behavior of D_A** (can be used as correctness proof later)
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow \mathcal{O}_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow \mathcal{O}_B(x')$ rejects $\Rightarrow D_A(x)$ rejects
- Suppose we picked "return same"
- Since we assumed the correctness of D_B, we must have $x' \in B$ if $\mathcal{O}_B(x')$ accepts, otherwise $x' \notin B$
- So we have
  - $x \in A \Rightarrow \cdots \Rightarrow x' \in B \Rightarrow \mathcal{O}_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow \cdots \Rightarrow x' \notin B \Rightarrow \mathcal{O}_B(x')$ rejects $\Rightarrow D_A(x)$ rejects

Input(s) For $\mathcal{O}_B$

- Desired Behavior of D_A: On input x:
  - $x \in A \Rightarrow x = a^n \Rightarrow \cdots \Rightarrow x = b^n \Rightarrow x' \in B \Rightarrow \mathcal{O}_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow \cdots \Rightarrow x \neq b^n \Rightarrow x' \notin B \Rightarrow \mathcal{O}_B(x')$ rejects $\Rightarrow D_A(x)$ rejects
- **Step 3: Generate input(s) for $\mathcal{O}_B$**
  - We want $x' = b^n$ if $x = a^n$
  - We want $x' \neq b^n$ if $x \neq a^n$
  - Note: Technically we can just check x (since we can hardcode 'a' into the machine), but since we want to use $\mathcal{O}_B$, we need to come out with the mapping
  - Brainstorm: What can we do to generate x'? Possibly by making use of x?
    - Flip all characters of x (aaa→bbb, aba→bab)

Putting Everything together

- Build D_A as follows:

  D_A = "On input x:

  $x' \leftarrow$Flip all bits in x

  Run $\mathcal{O}_B$ on x'

  if $\mathcal{O}_B(x')$ accepts **then accept**

  else reject"

- Or equivalently,

  Run $\mathcal{O}_B$ on x' and return the same"

- Correctness Proof:
  - $x \in A \Rightarrow x = a^n \Rightarrow x' = b^n \Rightarrow x' \in B \Rightarrow \mathcal{O}_B(x')$ accepts $\Rightarrow D_A(x)$ accepts
  - $x \notin A \Rightarrow x \neq a^n \Rightarrow x' \neq b^n \Rightarrow x' \notin B \Rightarrow \mathcal{O}_B(x')$ rejects $\Rightarrow D_A(x)$ rejects

Turing Reductions Overview

- Suppose we want to show that $A \leq_T B$
- **Step 1: Identify the inputs of D_A and $\mathcal{O}_B$**
  - Is the input a number? A string? Multiple strings? A machine?
- **Step 2: Draft Desired Behavior of D_A**
  - Choose between "return same" and "return opposite"
  - Return same: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \in B \Leftrightarrow \mathcal{O}_B(x')$ accepts $\Leftrightarrow D_A(x)$ accepts
  - Return opposite: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \notin B \Leftrightarrow \mathcal{O}_B(x')$ rejects $\Leftrightarrow D_A(x)$ accepts
- **Step 3: Generate input(s) for $\mathcal{O}_B$**
  - Return same: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \in B$ and $x \notin A \Rightarrow x' \notin B$?
  - Return opposite: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \notin B$ and $x \notin A \Rightarrow x' \in B$?

Note: Here we condense the two cases using iff

Proving Undecidability

Notes

Very Important Theorem

- Suppose $A \leq_T B$. If B is decidable, then A is decidable.
- Analogy: B is a *harder* problem than A. If I can solve the harder problem, then I can solve the easier problem.

- Contrapositive: $p = B$ is decidable; q = A is decidable
  - If p then q is equivalent to If $\neg q$ then $\neg p$
  - If A is <u>un</u>decidable, then B is <u>un</u>decidable

Known Fact	Conclusion
A decidable	?
A undecidable	B undecidable
B decidable	A decidable
B undecidable	?

Theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.

Undecidability Proof Outline

- Know: [insert undecidable language] is undecidable
- Task: Prove L is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L is decidable
  - …[something happen]…
  - Therefore, [insert undecidable language] is now decidable. Contradiction.

Theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.

Undecidability Proof Outline

- Know: L_{BARBER} is undecidable
- Task: Prove L_{ACC} is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L_{ACC} is decidable
  - …[something happen]…
  - Therefore, L_{BARBER} is now decidable. Contradiction.

Theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.

Undecidability Proof Outline

- Know: L_{BARBER} is undecidable
- Task: Prove L_{ACC} is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L_{ACC} is decidable
  - We have shown that $L_{BARBER} \leq_T L_{ACC}$
  - By this theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.
  - Since we have $L_{BARBER} \leq_T L_{ACC}$ and L_{ACC} is decidable by assumption
  - Therefore, L_{BARBER} is now decidable. Contradiction.

TL; DPA

- We discussed how to prove that a language is undecidable using Turing reduction
- In general, to show that a language is undecidable, reduce it from an undecidable language ($L_{BARBER}, L_{ACC}, L_{HALT}$)

Theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.

Concept Check

- Which of the following proves that L is <u>un</u>decidable? [Select all applies]
  - $\Sigma^* \leq_T L$
  - $L \leq_T \emptyset$
  - $L \leq_T L_{BARBER}$
  - $L_{HALT} \leq_T L$

More Turing Reductions

Notes

More Turing Reductions

- Another thing you are allowed to do is to create a machine (without running it) and pass it to the backbox decider
- For example, I want to use the black box decider D_E that decides
$$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

D_A= "On input $(\langle M \rangle, x)$:

    Construct M' as follows: TODO

    Run $\mathcal{O}_E$ on $\langle M' \rangle$ and return same"

- **Discuss:** Why do I have to create M'? Why can't I just use M?
  - The input of $\mathcal{O}_E$ must be a machine
  - We don't know $L(M)$, so we can't tell if $\mathcal{O}_E$ accepts/ rejects M

More Turing Reductions

- For example, I want to use the black box decider $\mathcal{O}_E$ that decides
$$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

```
D_A = "On input (⟨M⟩, x):
    Construct M′ as follows: TODO
    Run O_E on ⟨M′⟩ and return same"
```

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ ... $\Rightarrow \mathcal{O}_E(\langle M' \rangle)$ accepts $\Rightarrow D_A(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ ... $\Rightarrow \mathcal{O}_E(\langle M' \rangle)$ rejects $\Rightarrow D_A(\langle M \rangle, x)$ rejects

More Turing Reductions

- For example, I want to use the black box decider $\mathcal{O}_E$ that decides
$$L_E = \{\langle M\rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

D_A= "On input $(\langle M\rangle, x)$:

Construct M' as follows: TODO

Run $\mathcal{O}_E$ on $\langle M'\rangle$ and return same"

- Correctness proof draft
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ ... $\Rightarrow L(M') = \emptyset \Rightarrow \mathcal{O}_E(\langle M'\rangle)$ accepts $\Rightarrow D_A(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ ... $\Rightarrow L(M') \neq \emptyset \Rightarrow \mathcal{O}_E(\langle M'\rangle)$ rejects $\Rightarrow D_A(\langle M\rangle, x)$ rejects

Brainstorm Time

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ … $\Rightarrow L(M') = \emptyset \Rightarrow D_A(\langle M' \rangle)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ … $\Rightarrow L(M') \neq \emptyset \Rightarrow D_A(\langle M' \rangle)$ rejects
- **Brainstorm 1:** How to make $L(M') = \emptyset$ happen?
  - Just make M' rejects all inputs!
    - `M'= "On input w: Reject"`

Trigger this if M accepts x

- **Brainstorm 2:** How to make $L(M') \neq \emptyset$ happen?
  - Just make M' accepts *some* inputs, say "duck"
    - `M'= "On input w: If w = 'duck' then accept Else reject"`
  - Even easier: make M' accepts all inputs, i.e., $L(M') = \Sigma^*$
    - `M'= "On input w: accept"`

Trigger this if M does not accepts x

Putting Everything together…

D_A= "On input $(\langle M \rangle, x)$:

Construct M' as follows:

M' = "On input w:

Run M on x

if M accepts x **then reject**

else accept"

Run $\mathcal{O}_E$ on $\langle M' \rangle$ and return same"

- Correctness Proof:
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow M'$ rejects all inputs $\Rightarrow L(M') = \emptyset \Rightarrow \mathcal{O}_E(\langle M' \rangle)$ accepts $\Rightarrow D_A(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ M' accepts all inputs $\Rightarrow L(M') \neq \emptyset \Rightarrow \mathcal{O}_E(\langle M' \rangle)$ rejects $\Rightarrow D_A(\langle M \rangle, x)$ rejects

Turing Reduction Exercise

- Let L_U be defined as follows:

$$L_U = \{(\langle M_1 \rangle, \langle M_2 \rangle) : L(M_1) \cup L(M_2) = \emptyset\}$$

- Prove via reduction $L_{ACC} \leq_T L_U$ that L_U is undecidable.
- **Step 1: Identify the inputs of D_{ACC} and $\mathcal{O}_U$**
  - D_{ACC} takes $(\langle M \rangle, x)$
  - $\mathcal{O}_U$ takes $(\langle M_1 \rangle, \langle M_2 \rangle)$
- **Step 2: Draft Desired Behavior of D_A**
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ accepts $\Rightarrow \cdots \Rightarrow D_{ACC}(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ does not accept $\Rightarrow \cdots \Rightarrow D_{ACC}(\langle M \rangle, x)$ rejects

Suppose we pick “return same”

Turing Reduction Exercise

- Let L_U be defined as follows:
$$L_U = \{(\langle M_1 \rangle, \langle M_2 \rangle): L(M_1) \cup L(M_2) = \emptyset\}$$
- Prove via reduction $L_{ACC} \leq_T L_U$ that L_U is undecidable.
- **Step 1: Identify the inputs of D_{ACC} and $\mathcal{O}_U$**
  - D_{ACC} takes $(\langle M \rangle, x)$
  - $\mathcal{O}_U$ takes $(\langle M_1 \rangle, \langle M_2 \rangle)$
- **Step 2: Draft Desired Behavior of D_A**
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ accepts $\Rightarrow \cdots \Rightarrow \mathcal{O}_U(\langle M_1 \rangle, \langle M_2 \rangle)$ accepts $\Rightarrow D_{ACC}(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ does not accept $\Rightarrow \cdots \Rightarrow \mathcal{O}_U(\langle M_1 \rangle, \langle M_2 \rangle)$ rejects $\Rightarrow D_{ACC}(\langle M \rangle, x)$ rejects

Q: What does it mean for $\mathcal{O}_U(\langle M_1 \rangle, \langle M_2 \rangle)$ to accept/ reject?

Turing Reduction Exercise

- Let L_U be defined as follows:
$$L_U = \{(\langle M_1\rangle, \langle M_2\rangle) : L(M_1) \cup L(M_2) = \emptyset\}$$
- Prove via reduction $L_{ACC} \leq_T L_U$ that L_U is undecidable.
- **Step 1: Identify the inputs of $\boldsymbol{D_{ACC}}$ and $\boldsymbol{\mathcal{O}_U}$**
  - D_{ACC} takes $(\langle M\rangle, x)$
  - $\mathcal{O}_U$ takes $(\langle M_1\rangle, \langle M_2\rangle)$
- **Step 2: Draft Desired Behavior of $\boldsymbol{D_A}$**
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M(x)$ accepts $\Rightarrow \cdots \Rightarrow L(M_1) \cup L(M_2) = \emptyset \Rightarrow \mathcal{O}_U(\langle M_1\rangle, \langle M_2\rangle)$ accepts $\Rightarrow D_{ACC}(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M(x)$ does not accept $\Rightarrow \cdots \Rightarrow L(M_1) \cup L(M_2) \neq \emptyset \Rightarrow \mathcal{O}_U(\langle M_1\rangle, \langle M_2\rangle)$ rejects $\Rightarrow D_{ACC}(\langle M\rangle, x)$ rejects

Turing Reduction Exercise

- Correctness proof draft
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M(x)$ accepts $\Rightarrow \cdots \Rightarrow L(M_1) \cup L(M_2) = \emptyset \Rightarrow \mathcal{O}_U(\langle M_1\rangle, \langle M_2\rangle)$ accepts $\Rightarrow D_{ACC}(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M(x)$ does not accept $\Rightarrow \cdots \Rightarrow L(M_1) \cup L(M_2) \neq \emptyset \Rightarrow \mathcal{O}_U(\langle M_1\rangle, \langle M_2\rangle)$ rejects $\Rightarrow D_{ACC}(\langle M\rangle, x)$ rejects
- **Brainstorm 1:** How to make $L(M_1) \cup L(M_2) = \emptyset$ happen?
  - Example: $\emptyset \cup \emptyset = \emptyset$

Trigger this if M accepts x

- **Brainstorm 2:** How to make $L(M_1) \cup L(M_2) \neq \emptyset$ happen?
  - Example: $\Sigma^* \cup \Sigma^* = \Sigma^* \neq \emptyset$

Trigger this if M does not accepts x

- Observe that we can use the same TM for M_1 and M_2. Why?
  - $L(M_1) = L(M_2)$ in both cases

Turing Reduction Exercise

D_{ACC}= "On input $(\langle M \rangle, x)$:

Construct M as follows:

M' = "On input w:

Run M on x

if M accepts x **then reject**

else accept"

Run $\mathcal{O}_U$ on $(\langle M' \rangle, \langle M' \rangle)$ and return same"

- $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ accepts $\Rightarrow L(M_1) = L(M_2) = \emptyset \Rightarrow L(M_1) \cup L(M_2) = \emptyset \Rightarrow \mathcal{O}_U(\langle M_1 \rangle, \langle M_2 \rangle)$ accepts $\Rightarrow D_{ACC}(\langle M \rangle, x)$ accepts
- $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M(x)$ does not accept $\Rightarrow L(M_1) = L(M_2) = \Sigma^* \Rightarrow L(M_1) \cup L(M_2) \neq \emptyset \Rightarrow \mathcal{O}_U(\langle M_1 \rangle, \langle M_2 \rangle)$ rejects $\Rightarrow D_{ACC}(\langle M \rangle, x)$ rejects

Make your own TMs

Make your own TMs

- Create a TM M such that
  - $L(M) = \Sigma^*$
  - $L(M) = \emptyset$
  - $L(M) = \{3,7,6\}$
  - $L(M)$ is finite
  - $|L(M)|$ is even
  - M loops on x if $x \notin \{3,7,6\}$?
- Create two TMs M_1 and M_2 such that
  - $L(M_1) \cup L(M_2) = \Sigma^*$
  - $L(M_1) \cup L(M_2) = \emptyset$
  - $L(M_1) \cap L(M_2) = \emptyset$
  - $|L(M_1) \cap L(M_2)| = 1$

M: M is a Turing machine and $L(M) = \Sigma^*$

- Example solution:

  M = "On input w:

  Ignore w

  accept"

M: M is a Turing machine and $L(M) = \emptyset$

- Example solution:

  M = "On input w:

  Ignore w

  reject"

M: M is a Turing machine and $L(M) = \{3,7,6\}$

- Example solution:

  M = "On input w:

  if $w \in \{3,7,6\}$ **then accept**

  else **reject**"

M: M is a Turing machine and $L(M)$ is finite

- Example solution:

  M = "On input w:

  if w = "duck" **then accept**

  else reject"

- In this example, $L(M) = \{\text{"duck"}\}$, which is finite

M: M is a Turing machine and $L(M)$ is even

- Example solution:

  M = "On input w:

  if w = "duck" **or** w = "chicken" **then accept**

  reject"

- In this example, $L(M) = \{\text{"duck"}, \text{"chicken"}\}$, so $|L(M)| = 2$ which is even

M: M is a Turing machine M loops on x if $x \notin \{3,7,6\}$

- Example solution:

  M = "On input w:

  if $w \in \{3,7,6\}$ **then accept**

  else loop"

- Note: We don't have to explicitly describe how we want the machine to loop, it can be something simple like

  for $x = 1,2,\ldots$ **do**

  do nothing

$M_1, M_2 \colon L(M_1) \cup L(M_2) = \Sigma^*$

- There are a lot of ways to make the union of two languages Σ^*, for example, we can have
  - $\{\text{the set of strings that start with a } 1\} \cup \{\text{the set of strings that does not start with a } 1\}$
  - $L \cup \bar{L}$
  - $\Sigma^* \cup \{\varepsilon\}$
  - $\Sigma^* \cup \Sigma^*$
- We just need to build M_1, M_2 based on the desired language. For example, say we want $L(M_1) = \Sigma^*$ and $L(M_2) = \{\varepsilon\}$,

M_1 = "On input w:

Ignore w

accept"

M_2 = "On input w:

if $w = \varepsilon$ **then accept**

else reject"

$M_1, M_2 : L(M_1) \cup L(M_2) = \emptyset$

- Example solution: $\emptyset \cup \emptyset = \emptyset$

M_1 = "On input w:

Ignore w

reject"

M_2 = "On input w:

Ignore w

reject"

$M_1, M_2: L(M_1) \cap L(M_2) = \emptyset$

- Example: $\emptyset \cap \emptyset = \emptyset$, you could also get more creative by having $L(M_2) = \overline{L(M_1)}$, or $\{0\} \cap \{1\}$

M_1 = "On input w:

Ignore w

reject"

M_2 = "On input w:

Ignore w

reject"

$M_1, M_2 : |L(M_1) \cap L(M_2)| = 1$

- Example: $\{a, b, c\} \cap \{a, d, e\} = \{a\}$

M_1 = "On input w:

if $w \in \{a, b, c\}$ **then accept**

else reject"

M_2 = "On input w:

if $w \in \{a, d, e\}$ **then accept**

else **reject**"

Back Matter

Reducibility of Decidable Languages

- **Claim:** $L \leq_T L'$ for any decidable language L and any language L'.
- **Intuition:** Suppose L is decidable, there is a TM D that decides it. Given an oracle E that decides L', a machine that decides L given access to E is simply… itself!
- The point is this: In a Turing reduction from L to L', even though an oracle that decides L'is available, the reduction is not required to use it

$\leq_T$ is Reflexive

- **Claim:** Every language is Turing-reducible to itself, i.e., $L \leq_T L$ for all language L
- **Intuition:** If I have access to an oracle that decides L, then… I can already decide L!

$\leq_T$ is Transitive

- **Claim:** $\leq_T$ is transitive, i.e., if $A \leq_T B$ and $B \leq_T C$, then $A \leq_T C$
- **Proof:** By assumption, there is a TM M_A that could decide A with access to an oracle $\mathcal{O}_B$ that decides B. Similarly, there is a TM M_B that could decide B with access to an oracle $\mathcal{O}_C$that decides C.

This does not have to be the same oracle that M_B uses

- Now, we construct a decider D_A for A using some oracle $\mathcal{O}_C'$ that decides C as follow:
  - It does what M_A does, except whenever M_A would query its oracle $\mathcal{O}_B$ on some input, D_A instead runs M_B as a subroutine on the same input. Whenever M_B queries $\mathcal{O}_C$, M makes the same query to its own oracle $\mathcal{O}_C'$ and provides the answer to M_B
- **Analysis:** Since $\mathcal{O}_C'$ is a decider of C, it must give the same output as $\mathcal{O}_C$, which by assumption allows M_B to decide B. Hence, D_A has access to a correct decider of B, which by assumption allows D_A to decide A.
- Warning: $A \leq_T B$ and $B \leq_T A$ does NOT imply that $A = B$!

Existence of a Common Reducible Language

- **Claim:** For *any* two languages A and B, a language J exists, where $A \leq_T J$ and $B \leq_T J$
- Proof: Let $J = 0A \cup 1B$ (this is equivalent to what both math and CS folks often call a "disjoint union" or "tagged union"- intuitively it lets us union two sets while also keeping track of which set each element came from)
- Now, let $\mathcal{O}_J$ be an oracle that decides J.
- We can decide A using the decider D_A = "On input w, query the oracle about $0w$ and return the same."
- Similarly, we can decide B using the decider D_B = "On input w, query the oracle about $1w$ and return the same."

Turing Incomparable

- **Claim:** There exists two languages A and B such that $A \nleq_T B$ and $B \nleq_T A$
- In this statement, we don't have any Turing Machine that can work as Oracle Turing Machine to detect decidability and Turing reducibility.
- So, there is no language from A and B which could be replaced by any intermediate language.
- Also, Turing Machine cannot be constructed until decidability is proved.
- Thus, no solution can be defined in our case to prove decidability of two languages so that language A and B are not Turing Reducible.

Rice's Theorem

- Let A be a recognizable language, and define the language

$$L_A = \{\langle M \rangle : M \text{ is a Turing machine and } L(M) = A\}$$

- Consisting of (the description) of all Turing machines M for which $L(M) = A$. Then L_A is undecidable