Probability

- **Probability of events**: $P(\Omega) = 2^{|\Omega|}$, where $P(\Omega)$ is the set of all events
- **Uniform distribution**: Every outcome is the same: size of event divided by size of sample space Ω ($P(\text{event}) = \frac{|\text{event}|}{|\Omega|}$)
- **Non-uniform distribution**: Different outcomes (e.g. $P(\text{heads}) = \frac{1}{3}; P(\text{tails}) = \frac{2}{3}$)
- **Condition probability**: $P(A|B) = \frac{P(A \cap B)}{P(B)}$ ($P(A|B)$ not defined if $P(B) = 0$)
- **Independence**
  - A is <u>attracted</u> to B if $P(A|B) > P(A)$
  - A is <u>repelled</u> by B if $P(A|B) < P(A)$
  - A is <u>indifferent</u> to B if $P(A|B) = P(A)$
  - A and B are independent if $P(A \cap B) = P(A) \times P(B)$
  - **Mutual Independence**: (e.g. $k = 3 \rightarrow 3$ events: A_1, A_2, A_3)
    * $P(A_1 \cap A_2) = P(A_1) \times P(A_2)$
    * $P(A_1 \cap A_3) = P(A_1) \times P(A_3)$
    * $P(A_2 \cap A_3) = P(A_2) \times P(A_3)$
    * $P(A_1 \cap A_2 \cap A_3) = P(A_1) \times P(A_2) \times P(A_3)$
  - **Pairwise Independence**: (e.g. $k = 3 \rightarrow 3$ events: A_1, A_2, A_3)
    * $P(A_1 \cap A_2) = P(A_1) \times P(A_2)$
    * $P(A_1 \cap A_3) = P(A_1) \times P(A_3)$
    * $P(A_2 \cap A_3) = P(A_2) \times P(A_3)$
- **Expected Value**: $E[X] = \sum_{\sigma \in \Omega} P(\sigma) \times X(\sigma)$ (**Conditional Expectation**: $E[X|B] = \sum_{\sigma \in \Omega} P_B(\sigma) \times X(\sigma)$)
- **Variance**: $var(X) = E[X^2] - E[X]^2$
- **Standard Deviation**: $\sigma(X) = \sqrt{var(X)}$
- **Inequalities**
  - **Basic**: $P(X \geq a) \leq \frac{E[X]}{a}$
  - **Markov**: $a \in \mathbb{R}$ such that $a > 0$, $P(|X| \geq a) \leq \frac{E[|X|]}{a}$
  - **Chebyshev**: $a \in \mathbb{R}$ such that $a > 0$, $P(|X| \geq a) \leq \frac{E[X^2]}{a^2}$
  - **Cantelli**: $a \in \mathbb{R}$ such that $a > 0$, $P(X - E[X] \geq a) \leq \frac{var(X)}{a^2 + var(X)}$
  - **Chernoff**: $P(X \geq (1+\theta)pn) \leq e^{-\frac{\theta^2}{3}pn}$

Let $A, B \subseteq \Sigma^\star$ for $\Sigma = \{a, b, c\}$, and let $x_{Yes}, x_{No} \in \Sigma^\star$, such that the following properties are satisfied.

(i) $B = \{\mu \in \Sigma^\star \mid \text{either } \mu \in A \text{ or the length of } \mu \text{ is even (or both)}\}$.

(ii) B is ***unrecognizable***.

(iii) $x_{Yes} \in A$ and $x_{No} \notin A$.

You were asked to prove that A is ***unrecognizable*** as well.

Solution: Consider the function $f : \Sigma^\star \to \Sigma^\star$ such that, for every string $\omega \in \Sigma^\star$,

$$f(\omega) = \begin{cases} x_{Yes} & \text{if the length of } \omega \text{ is even,} \\ \omega & \text{if the length of } \omega \text{ is odd.} \end{cases}$$

This function is defined on every string in $\Sigma^\star$, so that it is a ***total*** function from $\Sigma^\star$ to $\Sigma^\star$.

Claim #1: For every string $\omega \in \Sigma^\star$, if $\omega \in B$ then $f(\omega) \in A$.

Proof. Let $\omega \in \Sigma^\star$ such that $\omega \in B$. Then either the length of ω is even, or the length of ω is odd.

- If the length of ω is even then $f(\omega) = x_{Yes}$, and $x_{Yes} \in A$, so that $f(\omega) \in A$ in this case.
- If the length of ω is odd then, since $\omega \in B$ and the length of ω is *not* even, it follows by the definition of B (at line (i), above) that ω must belong to A.

  Now $f(\omega) = \omega$ when the length of ω is odd, so that $f(\omega) \in A$ in this case too.

Since $f(\omega) \in A$ in every possible case, this establishes the claim. □

Claim #2: For every string $\omega \in \Sigma^\star$, if $\omega \notin B$ then $f(\omega) \notin A$.

```
On input ω ∈ Σ* {
1.  if (the length of ω is even) {
2.      return x_Yes
    } else {
3.      return ω
    }
}
```

Figure 1: An Algorithm to Compute the Function f

Proof. Let $\omega \in \Sigma^\star$ such that $\omega \notin B$. Then it follows by the definition of B (at line (i), above) that $\omega \notin A$ and the length of ω is not even — so that the length of ω must be odd.

Since the length of ω is odd, $f(\omega) = \omega$ — so that $f(\omega) \notin A$ since $\omega \notin A$, as noted above. □

Claim #3: The function f is computable.

Proof. It follows by the definition of f that the algorithm shown in Figure 1 computes the function f. It is therefore necessary, and sufficient, to show that this function can be computed by a Turing machine.

It is reasonably easy to describe a standard ***one-tape*** Turing machine that computes f. Suppose that this a Turing machine with tape alphabet

$$\Gamma = \{0, 1, \#, \sqcup\}.$$

- In order to implement the first step, the Turing machine should begin by reading the symbol $\sigma \in \{0, 1, \sqcup\}$ that is visible on the (leftmost) cell that is visible on the tape. It will use its finite control to remember which symbol $\sigma \in \{0, 1, \sqcup\}$ it read.
  - If the symbol was "$\sqcup$" (so that $\omega = \lambda$) then the Turing machine should replace the symbol on the tape with #, using a transition that would move the tape head ***left*** (so that the tape head does not actually move), changing to a state that begins the execution of the step at line 2.
  - Otherwise the Turing machine should replace the symbol on the tape with #, moving the tape head ***right***, and moving to a state corresponding to the fact that the number of input symbols read, so far, is *even*.

    While the symbol seen is not "$\sqcup$" the Turing machine should leave the symbol on the tape unchanged, moving the tape head ***right***. It should either change from a state indicating that the number of input symbols is *even* to a state indicating that the number of input symbols is *odd*, or vice-versa.

    When "blank" is seen the Turing machine should move the tape head ***left***, leaving the "$\sqcup$" unchanged. If the number of input symbols that it saw was *even* then it should change to a state that begins an implementation of the step at line 2. Otherwise (the number of input symbols seen was *odd*) it should change to a state that begins an implementation of the step at line 3.
- In order to implement the step at line 2 the Turing machine should move its tape head ***left***, as long as the symbol seen is in $\{0, 1, \sqcup\}$, replacing each seen with "$\sqcup$", until the copy of # at the leftmost cell is seen. It should then replace this with the first symbol in x_{Yes} (or "$\sqcup$" if $x_{Yes} = \lambda$) and move right, writing each of the symbols in x_{Yes} until this string is on the tape. Since this is a fixed string (whose length is a constant) it is easy to move the tape head back to the leftmost cell, so that x_{Yes} is being returned as output.
- In order to implement the step at line 3 the Turing machine should move its tape head ***left***, without changing the symbols on the tape, until the copy of "#" marking the leftmost cell is visible. The finite control can be used to restore the symbol on the tape that was overwritten by "#" as a transition moving ***left*** is being followed, so that ω is on the tape, and the tape head is at the leftmost cell, when the execution ends.

Thus the algorithm shown in Figure 1 can be implemented using a Turing machine. Since this algorithm compute the function f it follows that f is computable, as claimed. □

It follows by Claims #1–#3 that f is a many-one reduction from B to A, so that

$$B \preceq_M A.$$

Since B is unrecognizable, and the set of recognizable languages is closed under many-one reductions, it follows that A is also unrecognizable.

Notes:

- It is not necessary to describe a Turing machine in as much detail, as in the above proof of Claim #3, to receive full marks. However, at least *some* attempt to show that an algorithm to compute f, using a Turing machine, is required.
- It is also possible to use ***closure properties of the set of all recognizable languages*** to answer this question. However, you could only use closure properties for this set that were introduced in the lecture notes without proving that these closure properties are correct — so that it was almost certainly easier to answer this question correctly by giving a many-one reduction like the above one.
- Since the set of recognizable languages is ***not*** closed under oracle reductions, this problem ***cannot*** be solved by giving an oracle reduction from B to A.

Many-One Reductions

- A many-one reduction from L_1 to L_2 is a total function $f : \Sigma_1^\star \to \Sigma_2^\star$. Following properties must be satisfied:
  1. For every string $w \in \Sigma_1^\star, w \in L_1$ iff $f(w) \in L_2$
  2. f is computable
- Meaning, L_1 is many-one reducible to L_2 ($L_1 \preceq_m L_2$)
- **Example 1**: Suppose L_1 is undecidable and that $x_{yes}, x_{no} \in \Sigma_2^\star$ such that $x_{yes} \in L_2$ and $x_{no} \notin L_2$. Consider the total function $g : \Sigma_1^\star \to \Sigma_2^\star$

$$g(w) = \begin{cases} x_{yes} & \text{if} \quad w \in L_1 \\ x_{no} & \text{if} \quad w \notin L_1 \end{cases}$$

- **Answer**: Function g is **not** a many-one reduction from L_1 to L_2
  - Function g is not computable. Can be shown by proof of contradiction - that is, assuming that g *is* computable.
- **Example 2**: Suppose that next, $L_1 = \emptyset$ and $L_1 \neq \Sigma_1^\star$. Consider a total function $h : \Sigma_1^\star \to \Sigma_2^\star$ such that $h(w) = x_{yes}$
- **Answer**: Not a many-one reduction from L_1 to L_2
  - Since $L_1 \neq \Sigma_1^\star$ there exists a string $z \in \Sigma_1^\star$ such that $z \notin L_1$. However, $h(z) = x_{yes} \in L_2$, so the requirement "for all $w \in \Sigma_1^\star, w \in L_1$ iff $h(w) \in L_2$ is not satisfied.

DFA

(a) **DFA**

(b) **Describe set of strings corresponding to states**

- Prompt: $L = \{w \in \Sigma^\star | w \text{ ends with "ab" and the copies of "a" is even}\}$
- Format: $S_{\lambda,even} = \{w \in \Sigma^\star | w \text{ does not end with "a" or "ab", and copies of "a" is even}\}$ correponds to state q_0
- Do it for every state!

(c) **Claims needed to verify transitions out of start state**

- $\{w \cdot a | w \in S_0\} \subseteq S_{od}$
- $\{w \cdot b | w \in S_0\} \subseteq S_0$
- $\{w \cdot c | w \in S_0\} \subseteq S_0$

(d) **Proof that the transition out of the start state** (for some symbol) **is correct and well-defined** (e.g. b in $S_{\lambda,ev}$)

- Let $w \in S_{\lambda,ev}$ - so that w does not end with "a" or "ab" and the copies of "a" is even. Now, string $w \cdot b$ certainly cannot end with "a". In order for the string to end with "ab", w must end with "a".
- String $w \cdot b$ has as many copies of "a" as w does, so number of copies of "a" in $w \cdot b$ is even.

(e) **Additional claims** (format)

(i) Every string must belong to exactly **one** of the sets - that is, exactly one of (set of states here - e.g. S_0, S_a)

(ii) $\lambda \in S_0$ because it is the start state

(iii) Need to prove that $S \cap L = \emptyset$ for every set not in the set F of accepting states (e.g. $S_{\lambda,ev} \cap L = \emptyset$)

(iv) Need to prove that $S \subseteq L$ for every set that belongs to F (e.g. $S_{ab} \subseteq L$)