The first three questions on this assignment should be completed as group work. One copy of the solution to each problem should be submitted by noon on Wednesday or Thursday 4/22 or 4/23 depending on your group's meeting time.

The final question on this assignment should be submitted independently by each student by Tuesday 4/21. You may discuss approaches to the question with members of your working group, but the final writeup you submit should represent your own work in the same sense that was expected for all homework submissions during the first half of the semester.

1. Given an alphabet Σ and a string $w \in \Sigma^*$ and a symbol $x \in \Sigma$, let $\text{occurs}(x, w)$ equal the number of times the symbol x appears in w. We can define an infinite collection of languages

   $$L_{uni-\Sigma} = \{w \mid \text{ for all } x, y \in \Sigma, \text{occurs}(x, w) = \text{occurs}(y, w)\}$$

   That is $L_{uni-\Sigma}$ is the set of strings over a particular alphabet Σ in which all of the symbols in the alphabet occur equally often.

   Give a formal description of a family of Turing machines that decides $L_{uni-\Sigma}$ for an arbitrary alphabet Σ. Your formal description should include a precise description of the machine's state set and δ function written in terms of Σ. The formal description should be accompanied by a brief explanation of the intuition behind the machine (which may be very similar to the explanation given for the JFlap-based machine for $L_{uni-\{a,b,c\}}$ you submitted last week).

2. An unrestricted grammar is a quadruple $G = (V, \Sigma, R, S)$ where

   - V is a finite set of non-terminal symbols;
   - Σ is a finite set of terminal symbols disjoint from V;
   - $S \in V$ is the unique start symbol.
   - $R \subset (V \cup \Sigma)^*(V)(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ is a finite set of rules .

   The only difference between an unrestricted grammar and a context-free grammar is in the rules. Rules in context-free grammars have a single non-terminal on the left hand side, whereas rules in an unrestricted grammar may have any string of terminals and non-terminals on the left side, but must include at least one non-terminal. As with context-free grammars, although rules are formally tuples, they are conventionally written with an arrow separating the right and left sides.

   Derivations are similar to context-free grammars except the definition of "yields" is slightly different: we may substitute the right hand side of any rule into a derivation if the derivation has a substring matching the left hand side of the rule. More formally:

Yields Given an unrestricted grammar, $G = (V, \Sigma, S, R)$, and two strings x and y in $(V \cup \Sigma)^*$ such that $x = \alpha\sigma\beta$ and $y = \alpha\gamma\beta$ where $\alpha, \sigma, \gamma, \beta \in (V \cup \Sigma)^*$ and $(\sigma, \gamma) \in R$ we say that x *yields (or directly derives)* y. In this case we write

$$x \Longrightarrow y$$

For example, here's a grammar that generates the language $\{a^n b^n c^n \mid n \geq 1\}$:

$$\begin{array}{lclcl}
\text{S} & \rightarrow & \text{ABCS} & \mid & T_c \\
\text{CA} & \rightarrow & \text{AC} & & \\
\text{BA} & \rightarrow & \text{AB} & & \\
\text{CB} & \rightarrow & \text{BC} & & \\
\text{C}T_c & \rightarrow & T_c\text{c} & \mid & T_b\text{c} \\
\text{B}T_b & \rightarrow & T_b\text{b} & \mid & T_a\text{b} \\
\text{A}T_a & \rightarrow & T_a\text{a} & & \\
T_a & \rightarrow & \varepsilon & &
\end{array}$$

Let $L \subseteq \Sigma^*$ be a language accepted by a Turing machine M. Show that L can be generated by an unrestricted grammar G.

To solve this problem, you should explain how to construct an unrestricted grammar G given a description of a Turing machine M in such a way that $L(G) = L(M)$. A good first step in this process is to build a grammar that generates all sentential forms of the form $w\#Q_0\, w!$ where w is any string over the input alphabet of the machine, # and ! are markers, and Q_0 is a non-terminal corresponding to the start state of the Turing Machine. Then, you should add rules to the grammar that are structured so that the steps that can be taken using the rules of the grammar modify the part of the sentential form after the # in a way that each step in a derivation reflects one move that might be made by the Turing machine. Finally, the grammar should be defined so that if the non-terminal representing the accept state of the Turing machine ever appears on the right hand side of the #, the productions of the grammar allow derivation steps that erase all symbols to the right of and including the # ending with a derivation of w.

As both a hint to get you started and an example of the degree of detail I would like to see in your answers, the following fragment of a grammar generates all sentential forms $w\#Q_0\, w!$ for a given Σ.

Let $G = (V, \Gamma', S, R)$ where:

- $V = \{Start, Gen, Ret\} \cup \{Dup_x \mid x \in \Sigma\} \cup \{Q_0\}$
- $\Gamma' = \Sigma \cup \{\#, !\}$
- The set of rules R, includes:
  - $S \rightarrow Gen\ \#\ !$

- $Gen\ \# \rightarrow x\ \#\ Dup_x$, for all $x \in \Sigma$
- $Gen\ \# \rightarrow \#\ Q_0$
- $Dup_x\ y \rightarrow y\ Dup_x$, for all $x, y \in \Sigma$
- $Dup_x\ ! \rightarrow Ret\ x\ !$
- $x\ Ret \rightarrow Ret\ x$, for all $x \in \Sigma$
- $\#\ Ret \rightarrow Gen\#$

Intuitively, The non-terminal *Gen* allows the grammar to add any symbol, x to the right end of the first copy of w. Doing so introduces a copy of the non-terminal Dup_x which forces the grammar to add a copy of x to the end of the second copy of w. Once this second copy is added, *Ret* makes it possible to add another *Gen* right before the # that separates the two copies of w so that this process can be repeated. Eventually, when all the symbols of w are in place, *Gen* also makes it possible to complete the process by placing Q_0 after the central #. Note, as defined, Q_0 is a non-terminal rather than a terminal. Therefore, none of the sentential forms this grammar generates are actually sentences. In other words, $L(G) = \emptyset$. Your job is to add non-terminals and rules to the grammar so that the sentential form $w\#Q_0\ w\ !$ derives w iff $w \in L(M)$.

3. Let $C \subseteq \Sigma_C^*$ be a language. Prove that C is Turing-recognizable if and only if a Turing-decidable language $D \subseteq \Sigma_D^*$ exists such that

$$C = \{x \mid \text{there exists } n \text{ such that } b(n)\#x \in D\ \}.$$

where $b(n)$ represents a string encoding the number n in binary with no leading zeroes and # is a delimiter symbol such that $\# \notin \Sigma_C$.

You should think of n as acting as a *measure* of the work required to show that x is accepted by C.

4. Alan Turing clearly never used any word processing software! Imagine if when you typed a character in your favorite editor, the new character replaced the character after the cursor rather than being added to the document between the characters before and after the cursor position. To add a single character to the middle of a document you would have to retype every symbol from the insertion point to the end of your document. This is exactly how standard Turing machines work. Inserting an extra symbol in the middle of the tape requires copying every single character that follows on the tape.

On the other hand, when using a modern text editor, you can either insert a new character, delete a character by pressing the delete key or move the cursor left or right using arrow keys without replacing any characters. So, imagine a variant of the Turing machine where at each step, the machine could either insert any symbol from the tape alphabet, delete the symbol under the tape head, or move one position left or right along the symbols on the tape without actually changing the contents of the tape (where, like Sipser, we assume an attempt to move left past the leftmost symbol on the tape will leave the head on the same symbol).

For this problem, I would like you to provide a formal description of "Editor-like" Turing machines that behave as described above. Your answer should begin with a sentence like "An Editor-like Turing machine is a 7-tuple ..." It should include appropriate modifications of the four definitions of "Turing machine", "configuration", "yields" and "accepts" provided in class. If any of these definitions requires no change, just say so rather than repeating them.

Think carefully about how the head should be positioned after a character is inserted or deleted. There are many options here. When you insert a symbol, the new symbol could be inserted before or after the symbol at which the tape head is positioned. Similarly, while it seems obvious that when a symbol is deleted it should be the symbol that is currently under the tape head, there is a question of whether after the delete is completed, the head should move to the symbol that had been before or after the deleted symbol.

For insertions, your formalism should insert the new symbol before the symbol at the tape head and leave the tape head on the symbol that caused the insertion (that is, on the symbol that is now immediately to the right of the inserted symbol). For deletion, the machine should delete the symbol under the tape head. The tape head should then move to the symbol that had preceded the deleted symbol unless the machine just deleted the leftmost symbol on the tape. In that case, the head should move to the new first symbol on the tape.

To make it easy for you to copy any of the LaTeX source needed to explain your modifications, I have included the versions of my definitions from the lecture notes below:

> **Definition:** A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where
>
> Q is a finite set of states,
>
> Σ is a finite input alphabet (not containing the blank symbol),
>
> Γ is a finite tape alphabet which is a superset of Σ including the blank symbol,
>
> $\delta : Q \times \Gamma \to Q \times \Gamma \times \{Left, Right\}$ is the transition function,
>
> q_0 is the start state,
>
> q_{accept} is the accept state, and
>
> $q_{reject} \neq q_{accept}$ is the reject state.
>
> **Definition:** A **configuration** of a Turing machine is a triple (u, q, v) where $q \in Q$ is the current state, uv is the contents of the non-blank portion of the tape with u being the portion to the left of the current head position and u being the portion

from the symbol currently under the head to the end of the non-blank tape.

Definition: We say the configuration (u, q, av) **yields** configuration (u', q', v') for $q, q' \in Q, a \in \Gamma$, and $u, v, u', v' \in \Gamma^*$ if for some b and $c \in \Gamma$:

- $\delta(q, a) = (q', c, Left), u = u'b$, and $v' = bcv$, or
- $\delta(q, a) = (q', c, Right)$, $u' = uc$ and $v' = v$, or
- $\delta(q, a) = (q', c, Left), u = u' = \epsilon$, and $v' = cv$, or
- $\delta(q, a) = (q', c, Right)$, $u' = uc$, $v = \epsilon$, and $v' = \text{\_}$.

Definition: A TM **accepts** a string $w \in \Sigma^*$ if there is a sequence of configurations that begins with (ϵ, q_0, w) and ends in (w', q_accept, w'') for some $w', w'' \in \Gamma^*$ where each configuration yields the following configuration in the series.