trunc\texttt{ate} (ababba)=ababba

\[\text{If } w \text{ is a set of strings}\]
\[
\text{trunc\texttt{ate}}(L) = \{\text{trunc\texttt{ate}}(w) : w \in L\}
\]

We're overloading this function name

\[\text{trunc\texttt{ate}}(\{ababba, bab, aabab\})\]
\[= \{ababbb, ba, aaba\}\]

Given some unknown DFA \(L\), construct a DFA for \(\text{trunc\texttt{ate}}(L)\).

This means - describe a computer program that takes as input a DFA \(M\), and outputs a DFA \(M'\), so that if \(L\) is the language accepted by \(M\), then \(M'\) accepts \(\text{trunc\texttt{ate}}(L)\).
To do this problem, you need to be specific—someone should be able to write the code from your description.

To get started, make a random machine and think about what machine would accept \( \text{truncate}(L) \) (where \( L \) is the language accepted by).
Regular grammars

So far, we have been thinking about regular languages - by def'n a set of strings that is the set accepted by some DFA.

We have 2 ways so far to describe a regular lang - or NFA
- give a DFA for it
or - give a reg. exp. for it.

Today: Another way to describe languages in general (and specifically regular lang.)

A grammar a set of rules for generating a language that has a specific form (later)

E.g. \( P = \{a, b, 3\} \quad V = \{S, C, D\} \)
To generate strings using a grammar, I start with the "start symbol" S, and keep making substitutions until all my symbols are in T.

\[
S \rightarrow CbD \rightarrow CDbD \rightarrow bDbD \\
\quad \rightarrow bDCS_{ab}D \rightarrow baCS_{ab}D \\
\quad \rightarrow babS_{ab}D \rightarrow babyC_babD \\
\quad \rightarrow [bababbabba]
\]

The final result is "a string generated by the grammar"
The language generated by a grammar is the set of all strings that can be generated by the grammar.

**Formal definition:**

A grammar is a set \((V, \Gamma, P, S)\) where

- \(V\) is a finite set of symbols (variables)
- \(\Gamma\) is a finite set of alphabet (terminal symbols)

We assume \(V \cap \Gamma = \emptyset\)

- \(S \in V\) is the start symbol
- \(P \subseteq V^* \times (V \cup \Gamma)^*\) are the production rules.

\(V^* \backslash \{\lambda\}\)

not allowed to replace nothing with something.
Def: A string $w$ is produced by a grammar $G=(V,T,P,S)$ if

1. $w \in T^*$
2. There is some sequence of strings $v_0, \ldots, v_k \in (V\cup T)^*$ where
   
   - $v_0 = S$
   - $v_k = w$ for each $i$,
   
   and "I can get from $v_i$ to $v_{i+1}$* by following a rule in $P"$

   - $v_i = xyz$, $v_{i+1} = xy'z$, where $x, y, y', z \in (V\cup T)^*$, and $(y, y') \in P$

This is the formal way to say "replace $y$ by $y'$ to get from $v_i$ to $v_{i+1}$."
Some boring grammars:

- {a, b}^* generates all strings that alternated
- ab, babab, bababab,

- a is followed by its
- capital letters
- are in lower case

To allow strings to also end with a:

- add S → T to S = T a
Def: A grammar is right-linear if every production in P looks like

\[ A \rightarrow \text{abcdefg } B \]

- single variable
- string using
- the alphabet
- (no variables)
- (could be \( \lambda \))

or

\[ A \rightarrow \text{abcdef} \]

- single variable
- string w/ no variables (could be \( \lambda \))

Next time - explain the corresponding between right-linear grammars and NFA's.