1. A reduction can be a mapping into a special class of a more general problem.

\[ E_{TM} = \{ \langle M \rangle : L(M) = \emptyset \} \]
\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle : L(M_1) = L(M_2) \} \]

Then \( E_{TM} \leq_m EQ_{TM} \) via the mapping \( f(\langle M \rangle) = \langle M, M_0 \rangle \),
where \( M_0 = \begin{array}{c}
\text{accepting} \\
\text{on all} \\
\text{c.e.}
\end{array} \quad L(M_0) = \emptyset \), so \( \langle M, M_0 \rangle \in EQ_{TM} \implies (M) \in E_{TM} \).

The mapping \( f \) is computable since it simply "bolts on" the \( M_0 \) code.
Thus \( E_{TM} \leq_m EQ_{TM} \), and since \( E_{TM} \) is not c.e., neither is \( EQ_{TM} \).
In fact \( \text{ALL}_{TM} \leq_m EQ_{TM} \), using \( M_1 = \begin{array}{c}
\text{accepting} \\
\text{on all} \\
\text{c.e.}
\end{array} \) instead of \( M_0 \).

Since \( \text{ALL}_{TM} \) is neither c.e. nor co-c.e., so is \( EQ_{TM} \). Reductions:

2. Reductions that don't simply "bolt-on" or "drop-in" blocks of code.

**Theorem:** There are computable functions \( f_1 \) and \( f_2 \) that given any one-tape TM \( M \) produce context-free grammars \( G_1, G_2, \) and \( G_3 \)
\( f_1(M) = G_1, f_2(M) = \langle G_2, G_3 \rangle \) such that:

\[ L(M) = \emptyset \iff L(G_1) = \emptyset \iff L(G_2) \cap L(G_3) = \emptyset \]

\( \odot \) shows \( E_{TM} \leq_m \text{ALL}_{EFK} \), and \( \circ \) reduces \( E_{TM} \) to "Does \( L(G_1) \cap L(G_2) = \emptyset \) ?"
Proof (sketch): We can write configurations (aka IP's) of a 1-type
TMM M in a format specified by the following regular expression.
\[ \tau = \left( \Gamma^* \cdot Q \cdot \Gamma \cdot \Gamma^* \right) \]
\( \Gamma \) is the start state.
\( Q \) is the input \( Q \) set.
\( \Gamma^* \) is the rest of the tape is blank.

Sequences of IP's have the regular format \( \tau^+ \) (one or more IP's).
The machine M defines a condition for an IP \( T \) to follow an
IP \( T \)' by one step of \( M \) as coded by an instruction in \( S \).

\[ \left[ u \# c \# v \right] \left[ u' \# c' \# v' \right] \]

If the move \( (q, c/d, S/r) \) then \( u' = u \) and \( v' = v \), with \( c' = d \).
If the move \( (q, c/d, R/r) \) then \( u' = Ud \), \( c' = \text{first char of } r \), \( v' = \text{rest of } v \).
If the move is \( (q, c/d, L/r) \), similar (or else, if \( v' = \epsilon \), \( c' = w \)).

This condition is like DOUBLEWORD (with \# markers).
If we write \( T \) backwards, it becomes like PALINDROME, with \#.

Define \( VHM = \{ T_0 T_1 T_2 \cdots T_t \in L(\tau^+) : \text{To is the start IP on some input } S, \text{ it is on accepting IP, and for all } \}
\( j, 1 \leq j \leq t, \text{ we may follow } I_{j-1} \}

\( VHM \) is like \( ^* \Delta \cdot \text{(DOUBLEWORD)} \cdot ^* \), and since CFLs are closed
under \( \cdot \), it is a CFL. \( f_t(M) = \text{a CFG } G_t \) for \( VHM \).
Also define \( VHR_m = \sum I_0 \cdot I_4 \cdot I_2 \cdot I_3 \cdot I_5 \cdot I_6 \). It is accepting and reversible if \( m \) is odd.

\( VHR \) is like (MARKEDPAL) \(^+\) handles \( I_1 \) follows \( I_2 \), \( V \) if \( I_5 \) \( \neq \) \( I_2 \) follows \( I_5 \) if \( I_7 \) follows \( I_5 \).

\( \wedge (\text{MARKEDPAL}) \). Handles the cold cases: \( I_1 \) followed by \( I_2 \)
\( I_2 \) followed by \( I_4 \) etc.

We can build CFGs \( G_2 \) and \( G_3 \) so \( VHR = L(G_2) \cap L(G_3) \).

\( M \in \mathcal{Fm} \iff VHR_m = \emptyset \iff L(G_2) \cap L(G_3) = \emptyset \iff f_2(<m>) = 0 \).

call it \( \text{ENC} \).

Last reduction in \( \delta \). \( V \) and \( VHR_m \) can both be decided by special TMs \( B \) that never go outside the cells initially occupied by their input \( \langle m \rangle \).

B is called a \underline{linear bounded automaton} \((LBA)\). Hence both \( \text{FO-LBA} \) and \( \text{ALL-LBA} \) are undecidable problems as well. This happens for ANY kind of deterministic machine that can verify proofs!

3. A little more in logic and "\( P \)" and "\( \overline{P} \)"

\( \text{SAT's feasibility} \): instance: A Boolean formula \( f(x_1, \ldots, x_n) \)

\[ \text{like } (x_1 \lor x_2) \land \neg (x_3 \land x_4) \lor (x_3 \land \overline{x}_1) \]

\( f \) \text{ satisfiable?} \iff \underline{Question: Is there an assignment } a_1, a_2, \ldots, a_n \in \{0, 1\} \text{ that makes } f(a_1, a_2, \ldots, a_n) = \text{true?} \]
**Defn 1:** A language $B$ belongs to $P$ if there is a deterministic TM $M$ such that $L(M) = B$, and for all inputs $x \in \Sigma^*$, $M(x)$ halts within $q(|x|)$ steps, where $q$ is a fixed polynomial function. That is, $t$ from $\text{VTM}$ is $\leq q(|x|)$.

**Defn 2:** $B$ belongs to $NP$ if there is a nondeterministic TM $N$ s.t. $L(N) = B$ and every branch of $N$'s computation halts within $q(n)$ steps, where $n = |x|$ and $q$ is some fixed polynomial.

**Examples:**
- All of our multi-type linear-time languages belong to $P$.
- All of our decision problems ultimately solved by BFS or BFA belong to $P$.
- With the EPEFG and ECFG algorithms.
- Any CFL belongs to $P$ (not trivially, indeed $A_{CFG} \notin P$ (not in textbook)).
- $\{ <f(x_1 \ldots x_n), a_1 a_2 \ldots a_n > | f(a_1 \ldots a_n) = \text{true} \}$
  
  This is value checking for logical formulas.

**However,** $\text{SAT} = \{ <f(x_1 \ldots x_n) > | (\exists a_1 \ldots a_n) f(a_1 \ldots a_n) = \text{true} \}$ is only known to belong to $NP$, via an NTM that on input $f$ guesses $a_1 \ldots a_n$ and then verifies $f(a_1 \ldots a_n) = \text{true}$.

**Theorem:** $P = NP \iff \text{SAT} \in P = \text{false}$, this is unknown.
A better "Cono Diagram" than I drew at the end of lecture:

Neither RE nor NP is closed under diagonalization.

**Diagram Language**

\[ C = NP \lor (\neg NP) \]

MULLER Llive here.

\[ \text{If } C = \text{NP} \lor (\neg \text{NP}) \]

\[ \text{is trickier; neither } C \text{ nor } (\neg C) \]

\[ \text{is NP, but its complement isn't.} \]

\[ \text{is a CFL, but its complement isn't.} \]

\[ \text{are not CFLs, but their complements are CFLs (as seen on HW).} \]

\[ \text{are not CFLs, but PAL is not a CFL.} \]

**Extra:** The main & last intended theoretical concepts that were missed are hardness and completeness under polynomial-time computable mapping reductions. We write \( A \leq_m^p B \) if \( A \leq_m B \) via a function \( f: \Sigma^* \rightarrow \Sigma^* \) such that \( y = f(x) \) is computable in time \( 1 \times 10^{10}(1) \), and of course \( x \in A \Rightarrow y \in B \) since \( f \) is a mapping reduction. Because the composition of two polynomials is a polynomial — like I said \( (n^3)^r = n^6 \) in lecture — we get the same kinds of theorems as in §5.3, but with \( P \) in place of \( DEC \) and \( NP \) in place of \( A \):s.

\[ \text{1. } B \in P \Rightarrow A \in P \]
\[ \text{2. } B \in NP \Rightarrow A \in NP \]
\[ \text{3. } B \in \text{co} NP \Rightarrow A \in \text{co} NP \]

\[ \text{Finally, } B \text{ is NP-hard if } A \leq_m^p B \text{ for all } A \in NP, \text{ and NP-complete if also } B \in NP. \text{ So the np-lown theorem states that SAT is NP-complete. Also ATM is complete for RE under } \leq_m^p. \]