Log Space Reducibility \( \leq \log_m p \) is finer than \( \leq_p \) since L \subseteq P.

Log-Space Computable Fns

Technical Point: If \( M \) and \( g \) belong to FL (function logspace) then so does \( g \circ f \).

Issue: If \( M \) computes \( y = f(x) \) and \( M' \) computes \( z = g(y) \), we can chain \( M \) and \( M' \) together, but we can't store \( y \) within the \( O(\log |x|) \) space.

- If the input tapes are right-only ("One-Way Logspace": L_1, FL_1) then the problem goes away: \( M' \) reads a bit of \( y \) as soon as \( M(x) \) outputs it, and since there is no re-reading of input, the bits of \( y \) need not be stored.

- If \( M \) is allowed a finite number \( r \) of left-to-right passes over \( x \), and \( M' \) is allowed a finite number \( s \) of left-to-right passes over \( y \), then \( z = g(y) = g(f(x)) \) can be computed with \( r \cdot s \) passes over \( x \), so OK.

"Finite Passes and (\log n)^{O(1)} Space" is likewise closed under composition. This is a popular definition of Streaming Algorithms.

General Case of Arbitrary Re-Reads: Instead of storing \( y \), the combined machine \( M'' \) stores the location \( i \) of the input head of \( M' \) on bit \( i \) of \( y \).

Whenever \( M' \) wants to move its input head left, \( M \) restarts from the beginning until it outputs bit \( i-1 \) of \( y \), which is stored.

If \( M' \) moves to \( i+1 \), it takes however long to output bit \( i+1 \). All the \( \log \) overhead is inefficient for \( y \) but stays within \( O(\log |x|) \) space.
Therefore \( \leq_m \) reduction are transitive: \( A \leq_m B \land B \leq_m C \Rightarrow A \leq_m C \).

In fact, every \( \leq_m \) and \( \leq^p_m \) reduction shown in the course has actually been a \( \leq_m \log \) reduction or even the sharper one-pass streaming kind.

**Hallmarks of a \( \leq_m \log \) Reduction:**

- The objects it constructs have an explicit formula. E.g.:
  \[
  G_\Phi = (V_\Phi, E_\Phi), \quad V_\Phi = \{ x_i; x_i: 1 \leq i \leq n \} \cup \{ x_i, \overline{x_i}; \text{ variable } x_i \text{ is in clause } C_i \}; \\
  E_\Phi = \{ \ldots \} \cup \{ \ldots \} \quad \text{etc. possibly negated.}
  \]

- The individual items used in building \( G_\Phi \) etc. are finite clumps of \( O(\log n) \)-sized labels such as variable numbers \( i \), clause \#s \( s \).

- (In consequence), local features of the target object \( G_\Phi \) (note) depend only on local features of the source object (e.g. \( G_\Phi \)) or on simple global connections—like copying \( \langle m, w \rangle \) or hooking up the \( B \) and \( C \) nodes in the 3SAT \( \leq_m \) \( G_3 \) example.

**Graph Accessibility Problem (GAP):**

- **INST:** A directed graph \( G \), nodes \( s \) \& \( t \in V(G) \).
- **QUES:** Is there a path from \( s \) to \( t \) in \( G \)?

**GAP \( \in \text{NL} \):** A log-space NTM can guess the path and verify it while storing only the current node.

**A \( \in \text{NL} \Rightarrow A \leq_m \log \) GAP:** Given an NTM \( N_A \) running in \( C \)-log space and build the ID graph \( G_x \) from Monday's lecture. Every ID \( \langle g, w, i_1, \ldots, i_k \rangle \) has size \( \leq \log |Q| + \log n + k \log n \approx \log |Q| \) \( \log \) size. Hence using \( \log n \)-space for any pair \( (S, I) \), our reduction function can cycle through all pairs and "stream out" those pairs giving \( I \downarrow_{N_A} J \) which are the edges of \( G_x \).

And we have \( S = I_0(s) = \langle s, \epsilon, 1-1 \rangle \) as the starting ID and \( t = I_f = \langle q_{final}, 1-1 \rangle \) as a unique accepting ID we can arrange by "good housekeeping." So \( x \in A \iff (G_x,s,t) \in C(A). \)