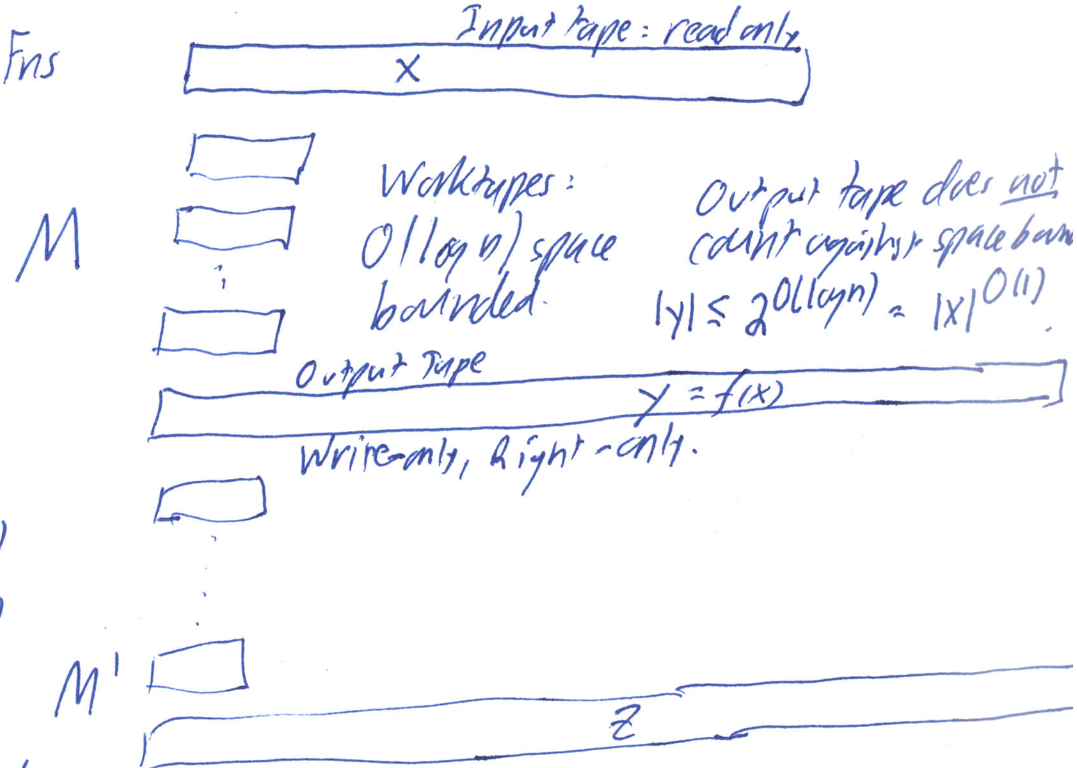
Log Space Reducibility $\leq_m^{\log}$ Finer than $\leq_m^{p}$ since $L \subseteq P$.

Log-Space Computable Fns

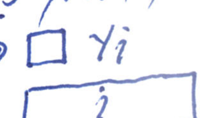


__Technical Point:__ If f and g belong to __FL__ (function logspace) then so does $g \circ f$. __Issue:__ If M computes $y = f(x)$ and M' computes $z = g(y)$ we can "chain" M and M' together, but we __can't store__ y, within the $O(\log |x|)$ space bd.

- 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 Algorithm__.

- General Case of Arbitrary Re-Reads: Instead of storing y, the combined machine M'' stores the location __i__ of the input head of M' and bit i of y. Whenever M' wants to move its input head __Left__, M __re-starts__ from the beginning until it outputs bit __i-1__ of y, which is stored



If M' moves to i+1, M takes however long to output bit i+1. All the re-starting is inefficient for __time__ but stays within O(log n) __space__.

Therefore $\le_m^{log}$ reductions are transitive: $A \le_m^{log} B \wedge B \le_m^{log} C \Rightarrow A \le_m^{log} C$.
In fact, every $\le_m$ and $\le_m^p$ reduction shown in the course has actually been a $\le_m^{log}$ reduction or even the sharper one-pass streaming kind.

Hallmarks of a $\le_m^{log}$ Reduction:

- The objects it constructs have an explicit formula. E.g.:
  $G_\phi = (V_\phi, E_\phi)$, $V_\phi = \{x_i, \bar{x}_i : 1 \le i \le n\} \cup \{x_{ij}, \bar{x}_{ij} :$ variable x_i is in clause C_j possibly negated$\}$
  $E_\phi = \{ \cdots \} \cup \{ \cdots \}$ etc.
- The individual items used in building G_ϕ etc. are finite clumps of $O(\log n)$-sized labels such as variable numbers i, clause #s j.
- (In consequence), local features of the target object G_ϕ (or etc.) depend only on local features of the source object (e.g., $C_1 \wedge \cdots \wedge C_m$) or on simple global connections—like copying $\langle M, w \rangle$ or hooking up the B and G nodes in the 3SAT $\le_m^{log}$ G3C 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 $\in$ NL: A $\log n$-space NTM can guess the path and verify it while storing only the current node.

$A \in NL \Rightarrow A \le_m^{log}$ GAP: Given an NTM N_A running in $c \cdot \log n$ space, and any x, build the ID graph G_x from Monday's lecture. Every ID $\langle q, w, i_1, \ldots, i_k \rangle$ has size $\approx \log|Q| + c\log n + k\log n$ ~~size~~ $O(\log n)$ size. Hence using $O(\log n)$ scratch space for any pair (I, J), our reduction function can cycle through all pairs and "stream out" those pairs giving $I \vdash_{N_A} J$ which are the edges of G_x. And we have $s = I_0(x) = \langle s, \epsilon, 1 \ldots 1 \rangle$ as the starting ID and $t = I_f = \langle q_{acc}, \text{"}1\text{"}, 1 \ldots 1 \rangle$ as a unique accepting ID we can arrange by "good housekeeping". So $x \in A \Leftrightarrow (G_x, s, t) \in$ GAP.