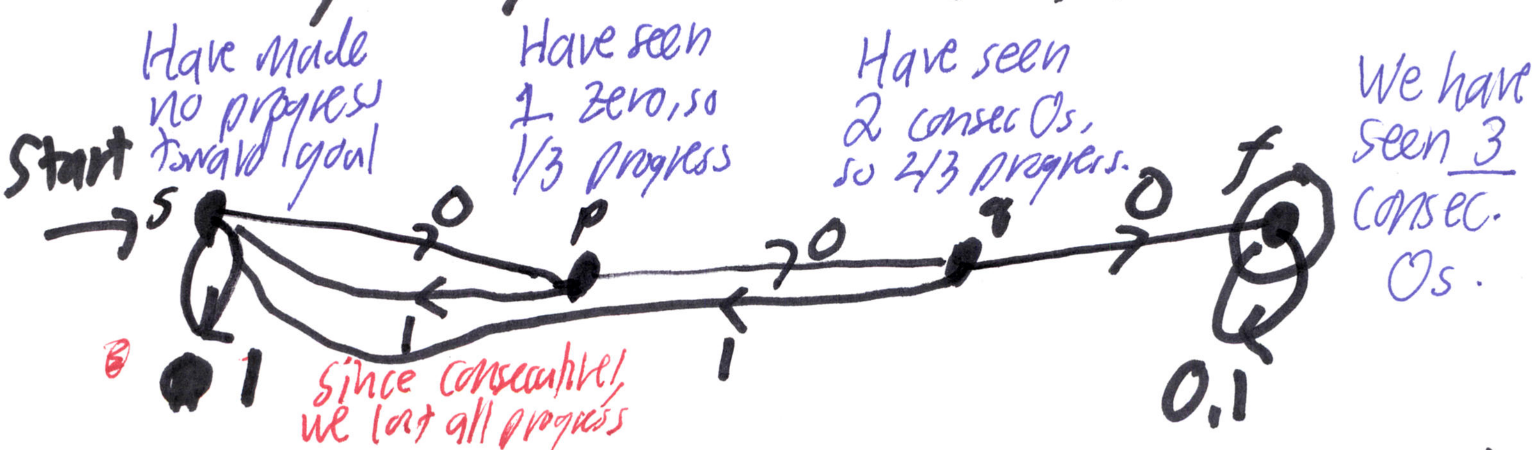
Let $A = \{x \in \{0,1\}^*$: x does not have 3 consecutive 0s in it$\}$.

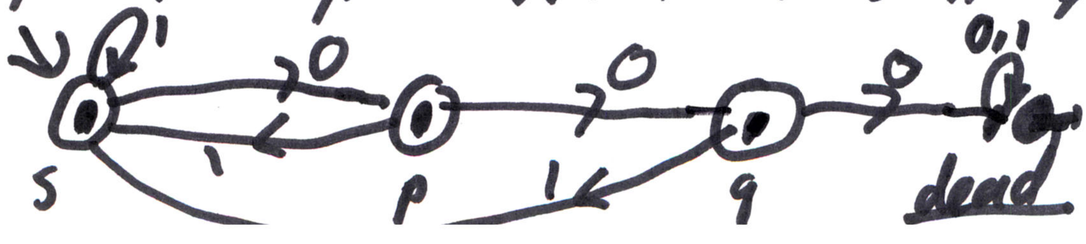
Build a DFA M such that $L(M) = A$.

The complement of A, which I denote by $\sim A$ or $\tilde{A}$ (text $\overline{A}$) is $\{x \in \{0,1\}^*$: x does have 3 consecutive 0s in it$\}$.

First design a "goal-oriented" DFA M' s.t. $L(M') = \tilde{A}$.



To get the original DFA M for $L(M) = A$, complement the accepting and rejecting states.



<u>Theorem</u>: Given any DFA $M = (Q, \Sigma, \delta, s, F)$ accepting a language A, we can build a DFA $\underline{M'} = (Q', \Sigma, \delta', s', F')$ such that $L(M') = \tilde{A}$.

<u>Proof</u>: Design M' by <u>taking</u> $Q' = Q$, $s' = s$, and $\delta' = \delta$ [i.e. states and arcs don't change].

But define $F' = \underbrace{Q \setminus F}_{\text{setminus}}$. Then

difference of sets.

Text uses an ordinary − sign.

$L(M') =_{def} \{x \in \Sigma^* :$ M' on input x ends up in a state in $F'\}$

$= \{x \in \Sigma^* :$ M' on x does not end up in a state in $\underline{F}$. $\}$

$= \sim \{x \in \Sigma^* :$ $M'(x)$ ends up in a state in $F\}$

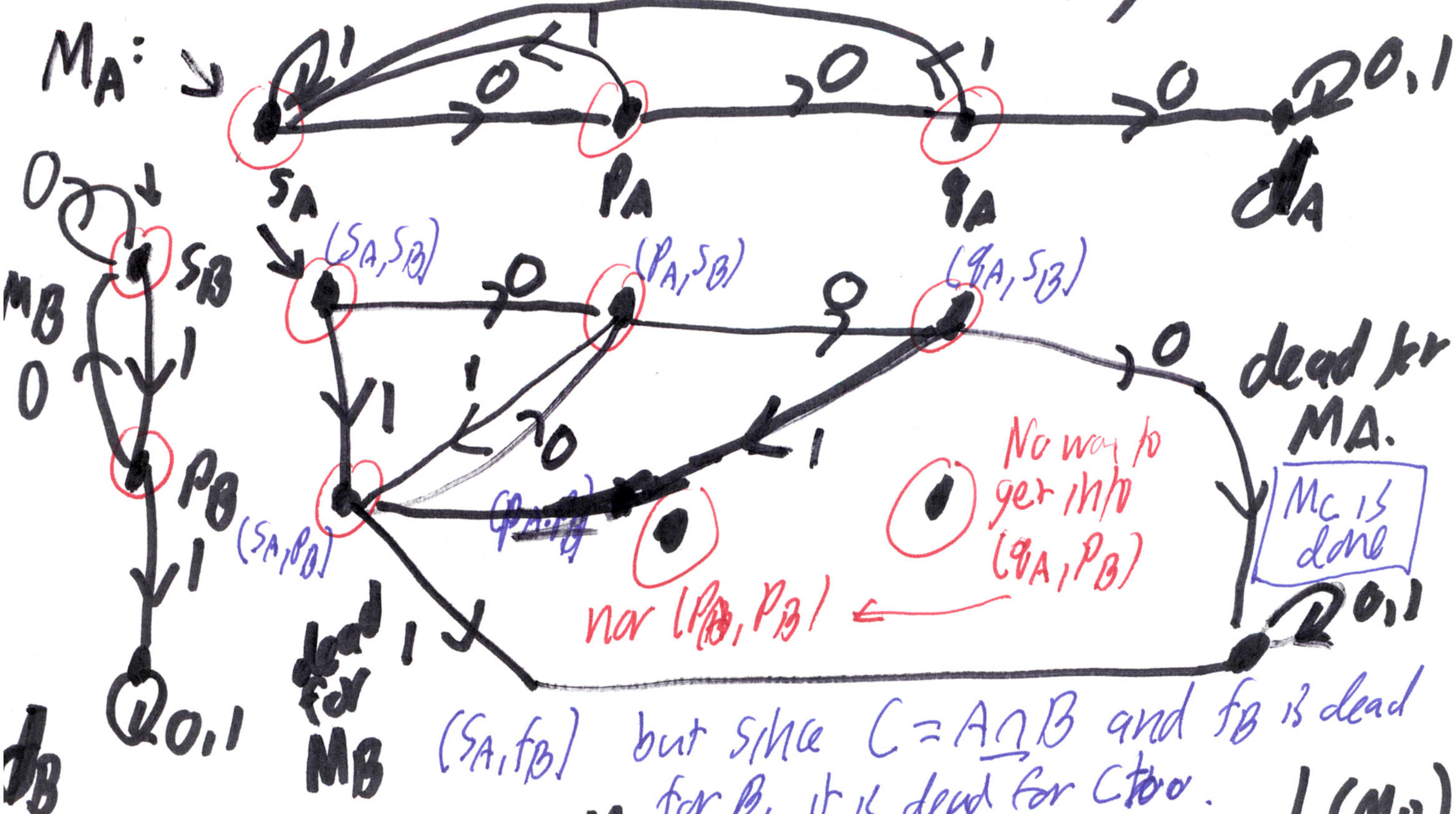
$= \sim L(M)$ since M and M' work the same.

$= \sim A$, since given that $L(M) = A$. $\boxtimes$

Cartesian Product Example:

$A = \{x : 000 \text{ is not a substring of } x\}$

$B = \{x : 11 \text{ is not a substring of } x\}$.



Design a DFA M_C such that $L(M_C) = L(M_A) \cap L(M_B)$.

<u>Theorem</u>: For any two languages A, B accepted by DFAs M_A and M_B, we can build a DFA M_C such that

$$L(M_C) = L(M_A) \cap L(M_B).$$

Since $\cap$ and $\sim$ generate all Boolean operations, we can get $A \cup B$, $A \triangle B$ etc. too.

Extra — was intended but will resume on Tue. (or in recitations)

Let **op** be any Boolean operation: $\cap, \cup, \setminus, \Delta$ etc.

Given DFAs $M_A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ and $M_B = (Q_B, \Sigma, \delta_B, s_B, F_B)$, design $M_C = (Q_C, \Sigma, \delta_C, s_C, F_C)$ by:

$Q_C = Q_A \times Q_B$ states of the form $q_C = (q_A, q_B)$

$s_C = (s_A, s_B)$

$$\delta_C((q_A, q_B), c) = (\delta_A(q_A, c), \delta_B(q_B, c))$$

Finally $F_C = \{(q_A, q_B) : \boxed{q_A \in F_A \text{ op } q_B \in F_B}\}$

Sets	Boolean	
$\cap$	$\wedge$	Can shortcut dead states
$\cup$	$\vee$	Can shortcut "nirvana states"
$\setminus B$	$a \wedge \neg b$	
Δ	$\oplus$, ie XOR	

So if op = XOR then

$F_C = \{(q_A, q_B) : [q_A \in F_A \wedge q_B \notin F_B]$ or $q_A \notin F_A \wedge q_B \in F_B\}$

(Breadth-first search from s_C can often save unnecessary work.)

Formal defn of $L(M)$ for DFAs (if time allows)
[lecture will do for NFAs]

<u>Defn</u>: A computation by a DFA $M=(Q,\Sigma,\delta,s,F)$ <u>on</u> a string $x\in\Sigma^*$ is a sequence

$$(q_0, x_1, q_1, x_2, q_2, \ldots, q_{n-1}, x_n, q_n)$$

such that $q_0 = s$,

for every i, $0 < i \le n$, $\delta(q_{i-1}, x_i) = q_i$.

The computation is <u>accepting</u> iff $q_n \in F$.

$L(M) \underset{\text{defined}}{=} \{x\in\Sigma^* : M$ has an accepting comp^n on input $x\}$.