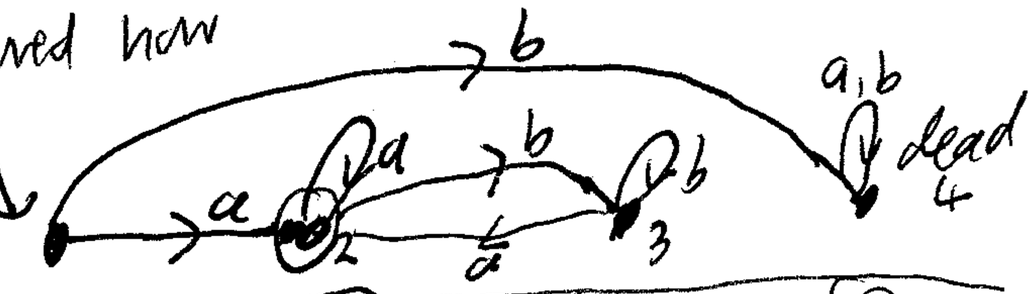
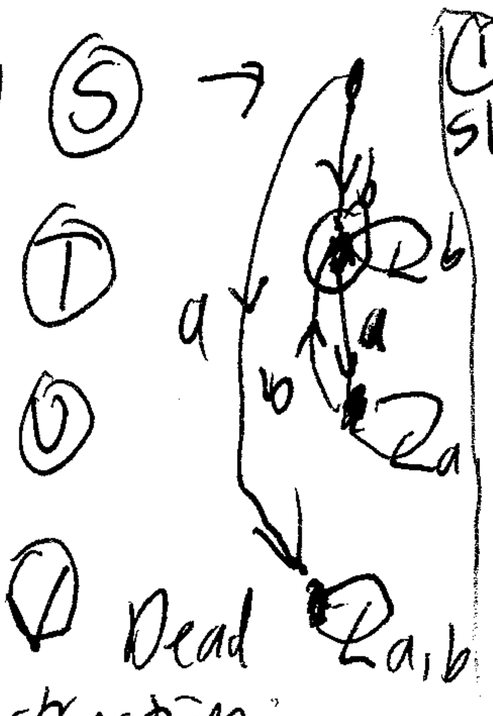
Lectures on Tue Feb 10 & Thu Feb 12.

[I first put comments on the states in the text's Example 1.11. Then I noted how it's language is the union of two languages having simpler DFAs M_1 and M_2. Then I showed how you could get the text's M_4 as a "Cartesian Product" of M_1 & M_2: $M_1 =$



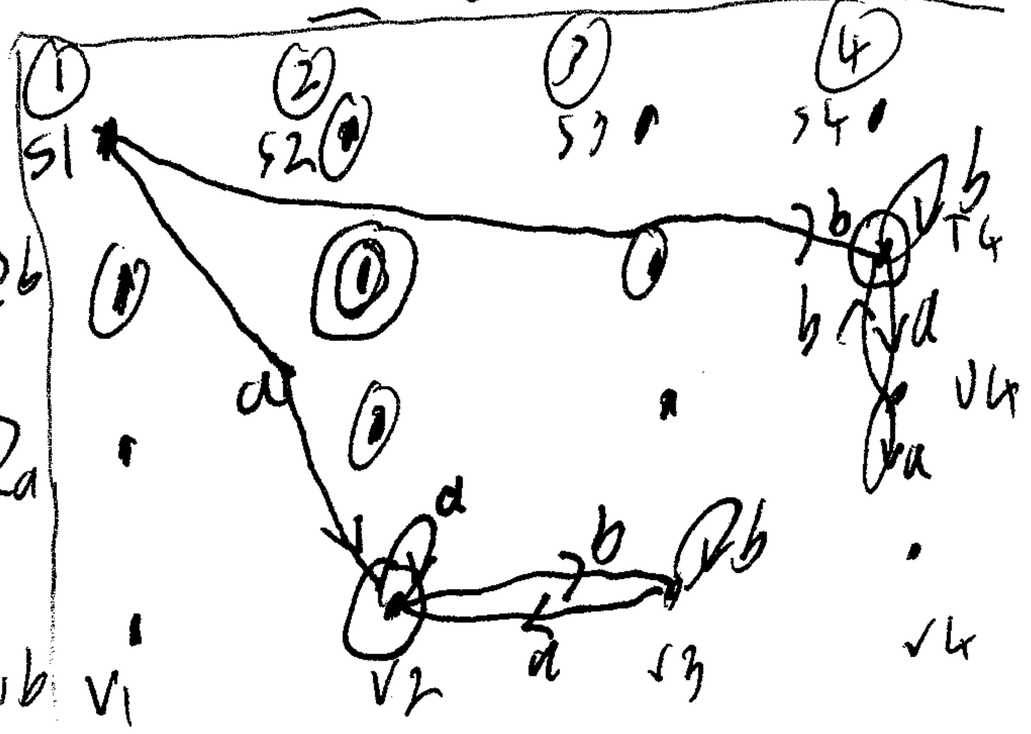
$L_2 = \{w : w$ begins and ends with $b\}$

$L = L_1 \cup L_2$ $M_2 =$



$Q_1 = \{1, 2, 3, 4\}$

$Q_2 = \{S, T, U, V\}$ ∅ Dead



Cartesian Product Construction:

$Q_3 = Q_1 \times Q_2 = \{1S, 1T, 1U, 1V, 2S, 2T, 2U, 2V, \ldots, 4V\}$ 16 states.

$\delta_3((P_1, P_2), c) = (\delta_1(P_1, c), \delta_2(P_2, c))$ $F_3 = \{(q_1, q_2) : q_1 \in F_1$ OR $q_2 \in F_2\}$

$F_3' = \{(q_1, q_2) : q_1 \in F_1$ AND $q_2 \in F_2\}$

$L' = L_1 \cap L_2 = \emptyset$ in this case.

Figure 1.14 shows the three-state machine M_5, which has a four-symbol input alphabet, $\Sigma = \{\langle\text{RESET}\rangle, 0, 1, 2\}$. We treat $\langle\text{RESET}\rangle$ as a single symbol.

State labels refer to count since last reset (or since start)

$\Sigma = \{0, 1, 2, \langle RESET \rangle\}$?

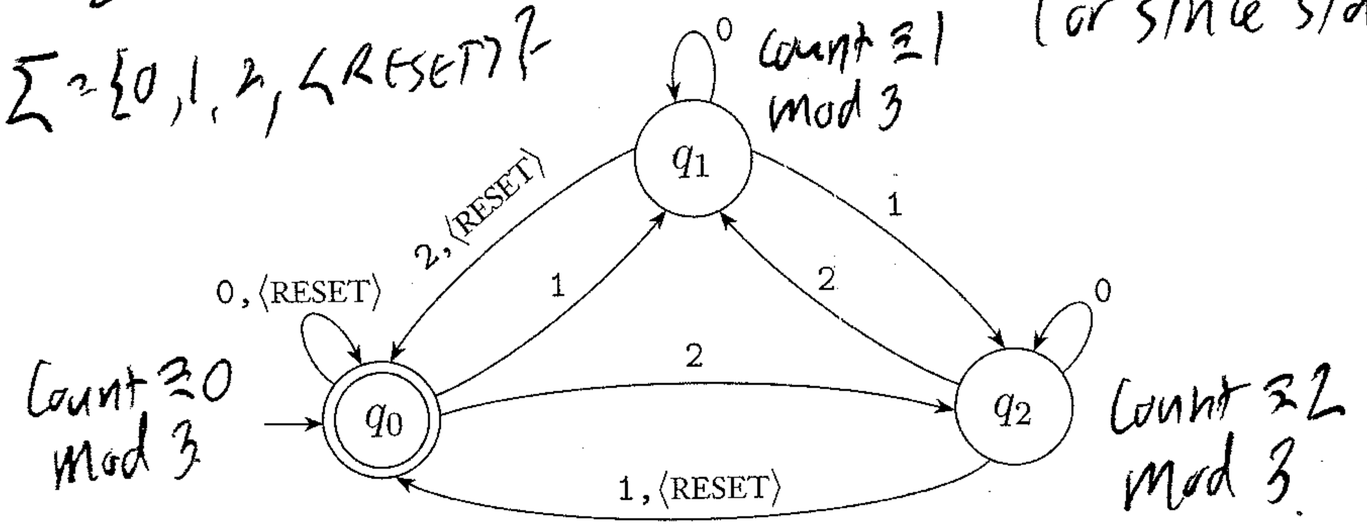
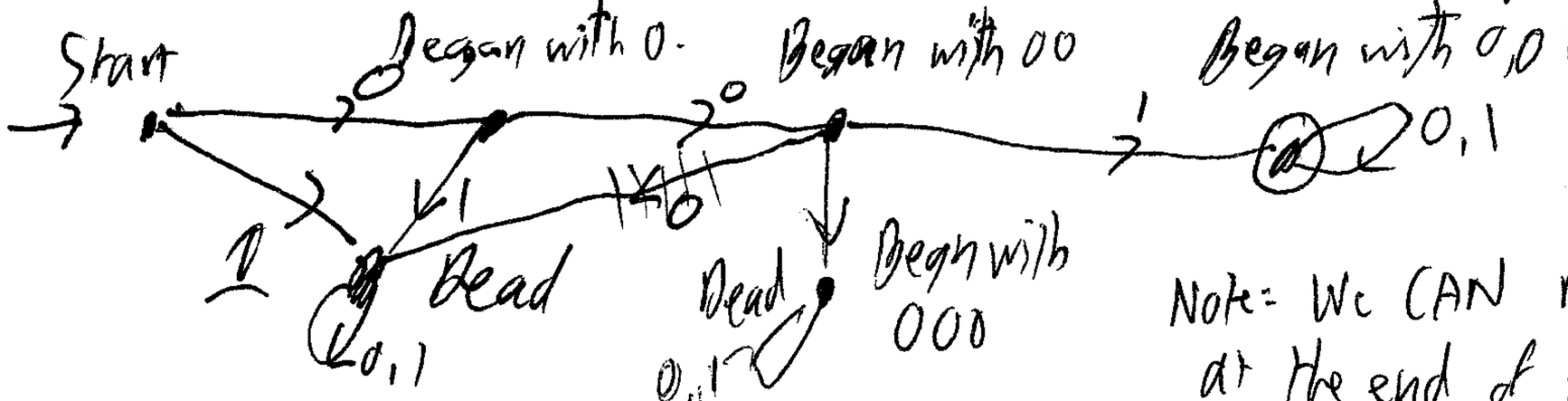


FIGURE 1.14
Finite automaton M_5

Machine M_5 keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives the $\langle\text{RESET}\rangle$ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3. Including ε !

$L_1 = \{w \in \{0,1\}^* : w$ <u>begins with</u> $001\}$.

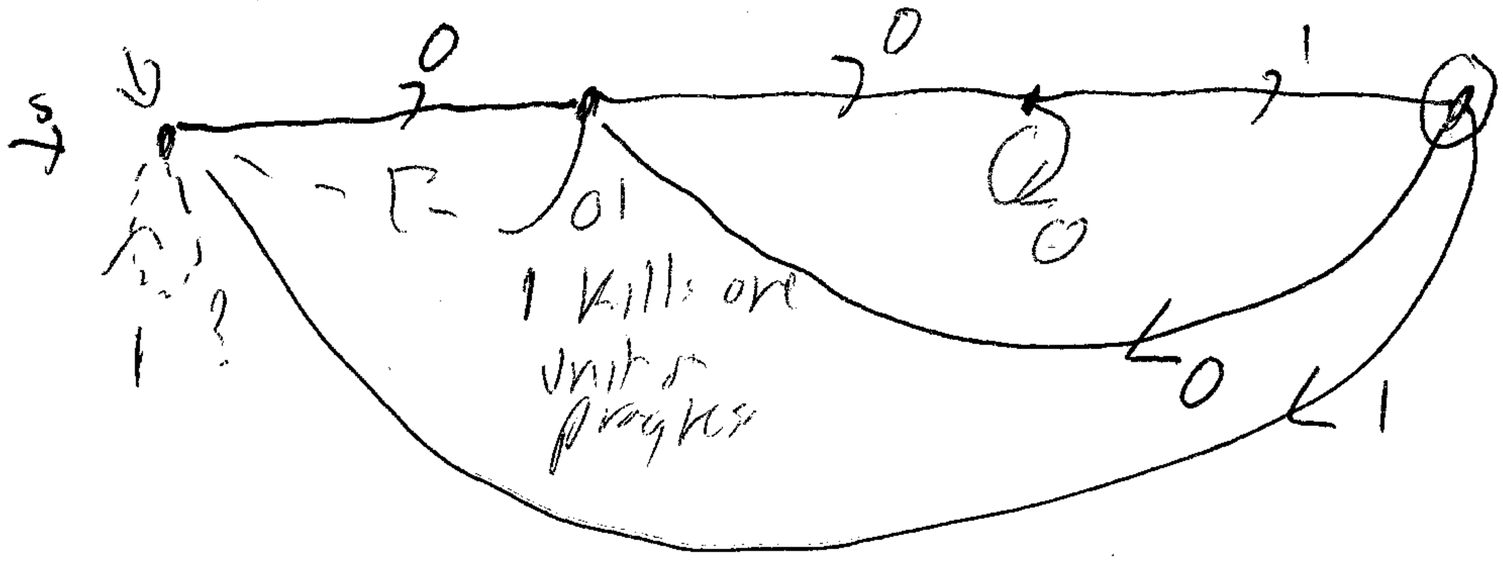


Note: We CAN recycle the diagram at the end of the last lecture.

$L_2 = \{w \in \{0,1\}^* : w$ ends with $001\}$.

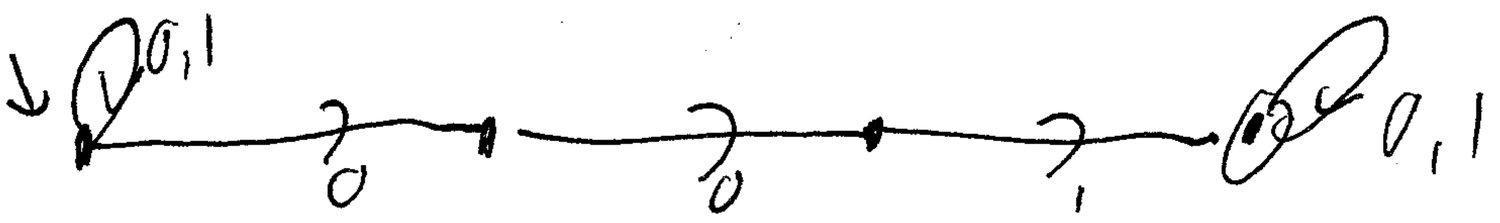
Idea: Maintain Progress Toward Desired state.

Zero Progress. Got the 0 Got 00 Last 3 chars 001



$L_3 = \{w \in \{0,1\}^* : w$ has 001 somewhere inside $\}$.

NFA



[The lecture plan intended to finish with $L_4 = \{w :$ 001 appears in w but not necessarily consecutive$\}$ There wasn't quite time, but this reappeared at the end on Thursday where the DFA made a nice punchline at the end.]

"Next time → formal definition of NFAs and idea of regular expressions.

Def^n: A nondeterministic finite automaton (NFA) is a 5-tuple $N = (Q, \Sigma, \delta, s, F)$ where: (like with a DFA)
- Q is a finite set of states
- Σ is a finite alphabet
- $s \in Q$ is the start state
- $F \subseteq Q$ is the set of accepting states

DFA had: $\delta : Q \times \Sigma \to Q$
which is the same as a relation
$\delta \subseteq (Q \times \Sigma) \times Q$ that happens to be a function.

"Ordinary NFA": $\delta \subseteq Q \times \Sigma \times Q$

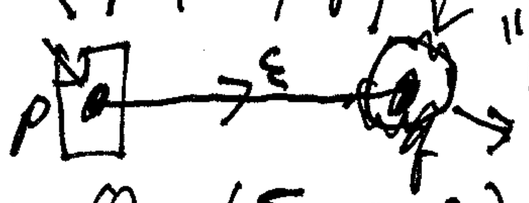
Not necessarily a function of the first two arguments. Can be partial or nondeterministic.

Typical instruction (p, c, q)

When graphed: 

$p = q$ is allowed: 

Side note: (p, ε, p) makes no sense "Whenever p, then also q for free."

Texts NFA adds a second instruction type: (p, ε, q) 

If $q \in F$, you may as well say $p \in F$ too.

Putting both types together: $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the full NFA instruction type.

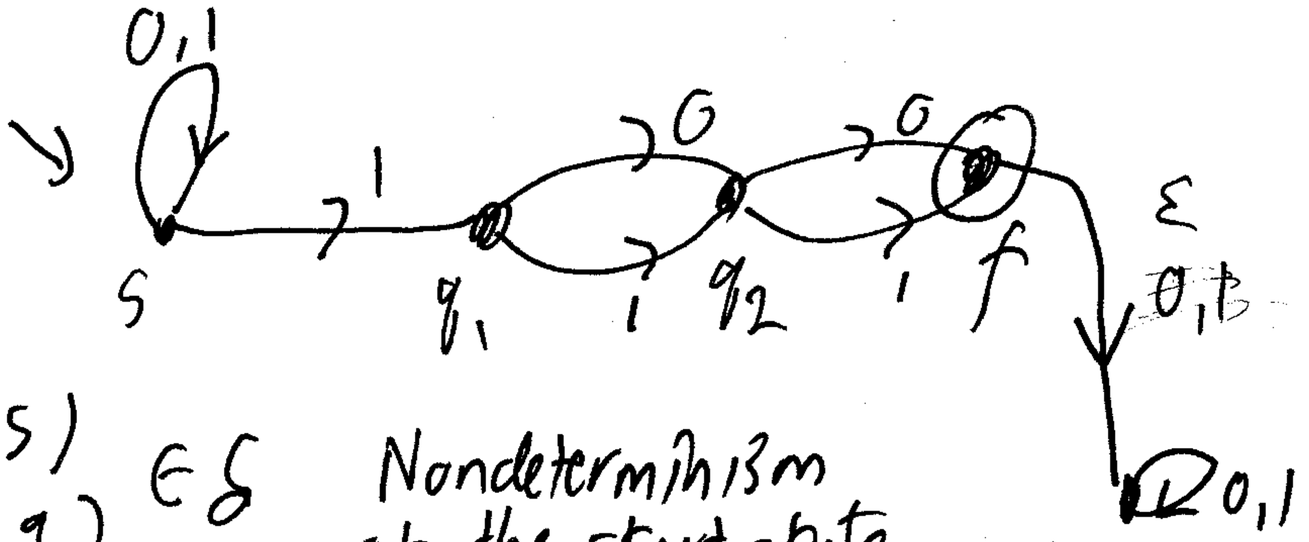
Def^n: A computation [path] for an NFA $N = (Q, \Sigma, \delta, s, F)$ is a sequence

$(r_0, y_1, r_1, y_2, r_2, \ldots, r_{m-2}, y_{m-1}, r_{m-1}, y_m, r_m)$ where for each i, $1 \le i \le m$,
(r_{i-1}, y_i, r_i) is an instruction in δ.

From the types, we can say:
- $r_0, r_1, \ldots, r_m$ are states in Q (of course not necessarily all distinct)
- Each y_i is either a char or $y_i = \varepsilon$.
- Either way $y_1 \cdot y_2 \cdots y_m$ "string together" to make a string $y \in \Sigma^*$.
  If there are no ε's, then $|y| = m$. But if there are k ε's, then $|y| = m - k$.
- We say that N can process y (entirely) from state r_0 to state r_m.
- ☆ When $r_0 = s$ and $r_m \in F$, then we have an accepting computation and $y \in L(N)$.
  Same defn is "inherited" by DFAs.

Examples With Their Computations

$N_3 =$



$(s, 1, s)$
$(s, 1, q_1)$ $\in \delta$ Nondeterminism at the start state.

There are no instructions $(f, 0, -)$. "Completely" Nor $(f, 1, -)$. Partial at f

We could "complete" the machine by filling in a dead state and transitions to it..

Consider $y = 110100$ Some computations:

$(s, 1, q_1, 1, q_2, 0, f)$ $m = 3$

Does this accept y? No: it does not process all of y. It does accept the prefix 110.

$(s, 1, s, 1, q_1, 0, q_2, 1, f$ Crash!

Neither of these counts as a valid computation on y, ie. processing y.

The completed NFA can at least process y like so:

$(s, 1, s, 1, q_1, 0, q_2, 1, f, 0, dead, 0, dead)$ $m = 6$, process 110100 (but does not accept it).

With ε instead, we can do

$(s, 1, s, 1, q_1, 0, q_2, 1, f, \varepsilon, dead, 0, dead, 0, dead)$. But, $m = 7$ steps.

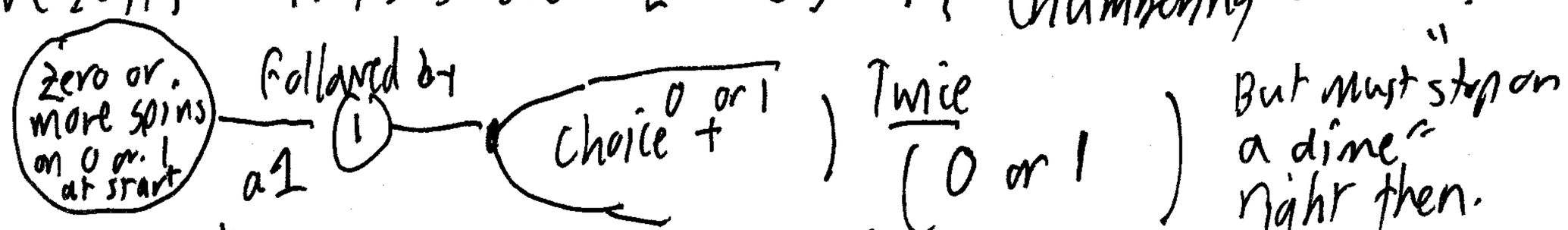
Impt: $1 \cdot 1 \cdot 0 \cdot 1 \cdot \varepsilon \cdot 0 \cdot 0$ still equals $y = 110100$.

Third computation does accept.

$(s, 1, s, 1, s, 0, s, 1, q_1, 0, q_2, 0, f)$ process all of $y = 110100$ So $y \in L(N)$.

$L(N) = \{w \in \{0,1\}^* : |w| \geq 3$ & $w[|w|-3] = 1\}$ (numbering from 0).

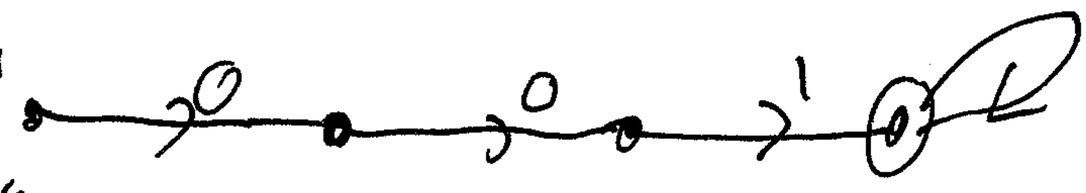
Idea of N and L(N):



* means zero or more times

$$(0+1)^* \cdot 1 \cdot (0+1) \cdot (0+1) = (0+1)^* 1 (0+1)^2.$$

$L_1 = \{w : w$ begins with $001\}$

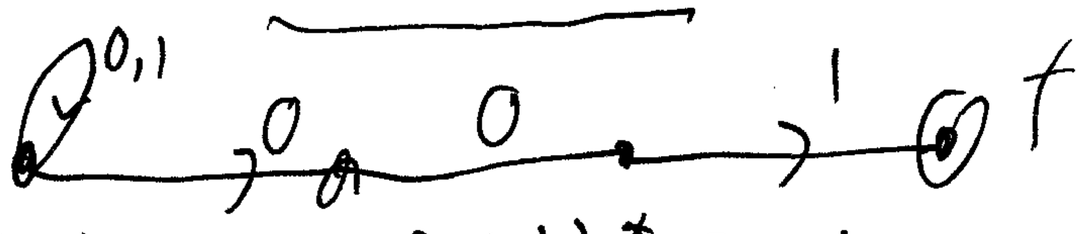
"DFA" 

Partial, hence
formally NFA N $\quad L(N) = 001 \cdot (0+1)^*$

set-theoretic. UNIX
Alt: $001(0 \cup 1)^*$ $001(0|1)^*$
UNIX $001(.*)$ $001[01]^*$
any chars. 0 or 1.
any chars in this list.

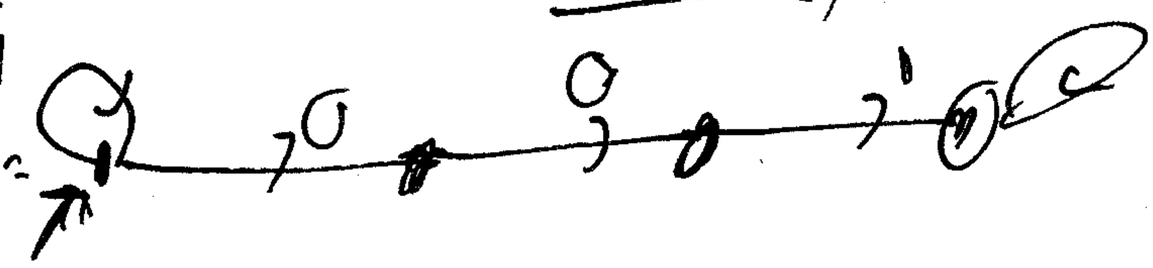
UNIX uses stars to mean "one or more" [without superscript]

$L_2 = \{w : w$ ends in $001\}$

NFA is easy N: 

Regexp for $L(N) = (0+1)^* 001$

$L_3 = \{w :$ W has the substring 001 somewhere inside it.$\}$

NFA: N_3: 

$L(N_3) = (0+1)^* 001 (0+1)^*$

$L_4 = \{w :$ W has 001 somewhere inside, not necessarily consecutive$\}$.
but following in that order.

$W =$ (Arbitrary) 0 (Arbitrary) 0 (Arbitrary) 1 (Arbitrary).

Regexp $R_4 = (0+1)^* 0 (0+1)^* 0 (0+1)^* 1 (0+1)^*$. Correct - but can we be more economical?

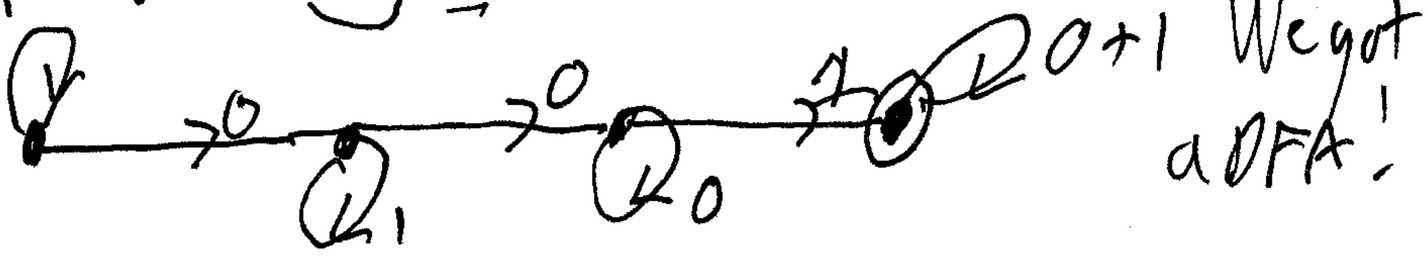
Alternative $\equiv$ $1^* 0 1^* 0 0^* 1 (0+1)^*$
R_5

Defn: Two regular expressions are equivalent if they denote the same language.
R_4 and R_5
In notation: $R_4 \equiv R_5$ means $L(R_4) = L(R_5)$. We can write "$R_4 = R_5$" when intent is clear.

We can also get equivalence by abbreviations. $00^* \equiv 0^+$

Superscript Plus means "one or more":

$R_6 = 1^* 0 1^* 0^+ 1 (0+1)^*$ $\quad = 00^*$

Design NFA from R_6: 

We got a DFA!