I first put comments on the slides in the text's Example 1.11. Then I noted how its language is the union of two languages having simpler DFAs $M_1$ and $M_2$. Then I showed how you could get the text $M_4$ as a "Cartesian product" of $M_1$ and $M_2$: $M_1 = M_2 \Rightarrow L_1 \cup L_2 = \emptyset$.

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

**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 (RESET) 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 0!
$L_1 = \{ \text{w begins with 01} \}$

Start: Began with 0 \hspace{1cm} Began with 00 \hspace{1cm} Began with 01

Dead \hspace{1cm} Dead \hspace{1cm} Begin with 00

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

$L_2 = \{ \text{w ends with 01} \}$

Zero progress \hspace{1cm} Go to 0 \hspace{1cm} Go to 00

Last 3 chars: 001

Make this a state after all.

$L_3 = \{ \text{w has 01 somewhere inside} \}$

NFA

[The lecture plan intended to finish with $L_4 = \{ \text{w: 001 appears in w, not necessarily consecutive} \}$

There wasn't quite time, but this reappeared at the end on Thursday when the DFA made a nice punchline at the end.]

""Next time: formal definition of NFAs and idea of regular expressions.\""
**Def.** A nondeterministic finite automaton (NFA) is a simple N = (Q, Σ, δ, s, F) where: Q is a finite set of states like with a DFA, Σ is a finite alphabet, s ∈ Q is the start state, F ⊆ Q is the set of accepting states.

An NFA has δ: Q × Σ → 2^Q which is the same as a relation $δ ⊆ Q × Σ × Q$ that happens to be a function.

**Def.** An ordinary NFA: \( δ \subseteq Q × Σ × Q \) is not necessarily a function of the first two arguments. Can be partial or nondeterministic.

Typical instruction when graphed:

- \( P \xrightarrow{c} Q \) \( P ≤ Q \) is allowed: \( P \xrightarrow{c} Q \xrightarrow{e} R \)

Text's NFA adds a second instruction type:

- \( P \xrightarrow{c, q} Q \) Side note: \( P, ε, P \) makes no sense.

If \( P ∈ E \), you may as well say \( P \xrightarrow{c} Q \) "Whenever \( P \), then also \( Q \) for free."

Putting both types together: \( δ \subseteq Q × (Σ ∪ ε) × Q \) is the full NFA instruction type.

**Def.** A computation (path) for an NFA N = (Q, Σ, δ, s, F) is a sequence \( (r_0, y_1, r_1, y_2, r_2, ..., r_{m-1}, y_{m-1}, r_{m-1}, y_m, r_m) \) where for each \( i \)

- \( 1 ≤ i ≤ m, \)
- \( (r_{i-1}, y_i, r_i) \) is an instruction in δ.

From the types, we can say:

- \( r_0, r_1, ..., r_m \) are states in Q (or cause not necessarily all distinct)
- Each \( y_i \) is either a char or \( y_i ∈ Σ^* \)
- Either way \( y_1, y_2, ..., y_m \) "string together" to make a string \( y ∈ Σ^* \)

- 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 \( r_m ∈ F \), then we have an accepting computation and \( y ∈ L(N) \).

Same defn is "inherited" by DFAs.
Examples with Their Computations

$N_3 = \begin{array}{c}
1 \\
1 \\
0, 1 \\
7 \\
0, 1 \\
1, 9_2 \\
5 \\
1, 9_1 \\
0, 1 \\
5 \\
1, 9_3 \\
0, 1 \\
\end{array}$

Non-determinism at the start state.
Dead

$(5, 1, 5) \in \Sigma$  Consider $y = 110100$  Some computations:

$(5, 1, 9, 1, 1, 9_2, 0, f)$  Does $this$ accept $y$?  No: it does not process all of $y$. It does accept the prefix $110$.

$(5, 1, 5, 1, 9, 0, 9_2, 1, f)$  Crash!  Neither of these count as a valid computation on $y$, i.e. processing $y$.

The completed NFA can at least process $y$ like so:

$(5, 1, 5, 1, 9, 0, 9_2, 0, f, 0, dead, 0, dead)$  $m = 6$, process $110100$ (but does not accept it).

With $\epsilon$ instead, we can do

$(5, 1, 5, 1, 9, 0, 9_2, 1, f, \epsilon, dead, 0, dead, 0, dead)$.  But, $m = 7$ steps.

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

Third computation does accept:

$(5, 1, 5, 1, 5, 0, 5, 1, 9, 0, 9_2, 0, f)$  process all of $y = 110100$

So $y \in L(N)$.

$L(N) = \{w \in \{0, 1\}^* : |w| \geq 3 \text{ and } w[1w-3] = 1^2 \}$ (numbering from 0).

Idea of $N$ and $L(N)$:

* means zero or more spins on $0$ or $1$ at start.

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

There are no instructions $f, 0, \_$.  Completely partial at $f$.  We could "complete" the machine by filling in a dead state and transitions to it.
\[ L_1 = \{ W : W \text{ begins with } 001 \} \]

("DFA"
[Diagram]
\[ 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \]

\[ L_2 = \{ W : W \text{ ends in } 001 \} \]

NFA is
[Diagram]
\[ 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \]

Regular for \( L_2 = \{ 0 (1)^* \} 001 \)

\[ L_3 = \{ W : W \text{ has the substring } 001 \text{ somewhere inside it} \} \]

NFA is
[Diagram]
\[ 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \]

\[ L_4 = \{ W : W \text{ has } 001 \text{ somewhere inside, not necessarily consecutive} \} \]

Regexp for \( L_4 = \{ (0+1)^* 0 (0+1)^* 0 (0+1)^* 1 (0+1)^* \} \)

Alternative: \( 1^* 0 1^* 0 0^* 1 (0+1)^* \)

Definition: Two regular expressions are equivalent if they denote the same language.

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. \( 00^* = 0^+ \)

\[ R_6 = 1^* 0 1^* 0^+ 1 (0+1)^* \]

Design NFA from \( R_6 \):
[Diagram]
\[ 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0^+ \]