One HW problem will be: Consider the "spears and dragons" DFA from the demo, which had alphabet \( \Sigma = \{ D, 0 \} \) where '0' means an empty room. Build deterministic finite automata that model the following alterations to the game:

- (a) You may hold up to 2 spears, but not 3. If you have 2 spears and kill a dragon, you are down to one spear.
- (b) You may hold up to 2 spears, but may only hold 2 spears for a limited time. If you get two empty rooms after picking up the second spear, you have to drop down to carrying one spear.

In each case, the language of your machine should be the set of strings \( \sigma \in \Sigma^* \) that represent "dungeons" where the Player survives. It is fine for the Player to exit holding zero spears, one spear, or two spears.

**Regular Expressions and Their Corresponding NFAs (with \( \varepsilon \)-transitions):**

We give an inductive definition with three base cases B1,B2,B3 and three inductive cases I1,I2,I3. At the same time, we give an inductive procedure for building an equivalent NFA with \( \varepsilon \)-transitions.

1. **(B1)** \( \emptyset \) is a regexp; \( L(\emptyset) = \emptyset \);
   \[ N_\emptyset = \begin{array}{c}
   s \\
   \varepsilon
   \end{array} \]
   \[ \delta = \emptyset \]

2. **(B2)** \( \varepsilon \) is a regexp; \( L(\varepsilon) = \{\varepsilon\} \);
   \[ N_\varepsilon = \begin{array}{c}
   s \\
   \varepsilon
   \end{array} \] \( \delta \) has \( (s, \varepsilon, f) \)

For all chars \( c \in \Sigma \):

3. **(B3)** \( c \) is a regexp; \( L(c) = \{c\} \);
   \[ N_c = \begin{array}{c}
   s \\
   c
   \end{array} \] \( \delta \) has \( (s, c, f) \)

This completes the basis of an inductive definition of regular expressions. Now let \( \alpha \) and \( \beta \) be any two regular expressions, with languages \( A = L(\alpha) \) and \( B = L(\beta) \). By inductive hypothesis (IH) we have NFAs \( N_\alpha \) and \( N_\beta \) such that \( L(N_\alpha) = A \) and \( L(N_\beta) = B \). Then:

4. **(I1)** \( \gamma = \alpha + \beta \) is a regexp; \( L(\gamma) = L(\alpha) \cup L(\beta) \).

Now to complete the induction case (I1) we need to show how to build an NFA \( \varepsilon \), \( N_\gamma \) such that \( L(N_\gamma) = L(\gamma) \). What we have to work with is (are) \( N_\alpha \) and \( N_\beta \). We know they have start states we can call \( s_\alpha \) and \( s_\beta \). Taking a cue from the base case NFAs, and mainly for convenience, we may suppose they have unique accepting states \( f_\alpha \) and \( f_\beta \). Besides that, we make no assumptions about their internal structure, so we draw them as "blobs":
The goal is to connect them together to make $N_\gamma$ with needed properties, also for the cases:

1. $\gamma = \alpha \cdot \beta$ is a regexp; $L(\gamma) = A \cdot B$.

2. $\gamma = \alpha^*$ is a regexp; $L(\gamma) = A^*$.

3. $\gamma = \alpha + \beta$ is a regexp; $L(\gamma) = L(\alpha) \cup L(\beta)$.

Construction for (I1):

$$N_\gamma = \epsilon$$

Note rule: $\epsilon \cdot x = x$

for all strings $x$.

This builds $N_\gamma$, but we still need to prove it is correct, i.e., $L(N_\gamma) = L(\gamma)$. Note the rhythm:

1. $L(N_\gamma) = L(N_\alpha) \cup L(N_\beta)$ by machine construction;
2. $L(N_\alpha) = L(\alpha)$ and $L(N_\beta) = L(\beta)$ by inductive hypothesis;
3. Thus $L(N_\gamma) = L(\alpha) \cup L(\beta) = L(\alpha + \beta) = L(\gamma)$ by definition of $\gamma$.

(II) $\gamma = \alpha \cdot \beta$ is a regexp; $L(\gamma) = A \cdot B = \{xy : x \in A \land y \in B\}$.

To write the reasoning out: $N_\gamma$ can process a string $z$ from its start state $s_\gamma = s_\alpha$ to its (unique) final state $f_\gamma = f_\beta$ if and only if $z$ has a first part $x$ that gets processed from $s_\alpha$ to $f_\alpha$ and a second part $y$ that gets processed from $s_\beta$ to $f_\beta$ (with the $\epsilon$ from $f_\alpha$ to $s_\beta$ silently in-between). I.e.: $z \in L(N_\gamma) \iff z \in \{x \cdot y : x \in L(N_\alpha) \land y \in L(N_\beta)\} \iff z \in L(N_\alpha) \cdot L(N_\beta)$. Thus $L(N_\gamma) = L(N_\alpha) \cdot L(N_\beta)$. By IH, this equals $L(\alpha) \cdot L(\beta)$, which by how the semantics of $\gamma = \alpha \cdot \beta$ is defined via $L(\gamma) = L(\alpha) \cdot L(\beta)$ finally gives us the needed conclusion $L(N_\gamma) = L(\gamma)$.

Now we finish the proof with the Kleene Star case---named for Stephen Kleene:
The Main Theorem About Regular Expressions and Finite Automata

**Theorem:** For any language $A$ over an alphabet $\Sigma$, the following statements are equivalent:

1. There is a regular expression $\alpha$ such that $A = L(\alpha)$.
2. There is an NFA $N$ such that $A = L(N)$.
3. There is a DFA $M$ such that $A = L(M)$.

Example: The "Leap of Faith" NFAs $N_k$ for any $k > 1$:

$$L(N_k) = (0 + 1)^*1(0 + 1)^{k-1} = \left\{ x \in \{0, 1\}^* : \text{the } k\text{th bit of } x \text{ from the end is a 1} \right\}.$$ 

**Fact** (will be proved later): Whereas the NFA $N_k$ has only $k + 1$ states, the smallest DFA $M_k$ such that $L(M_k) = L(N_k)$ requires $2^k$ states. This is a case of exponential blowup in the NFA-to-DFA algorithm. For now, we just care that an equivalent DFA can be built.