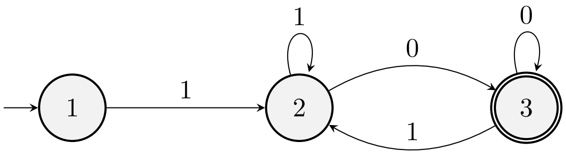
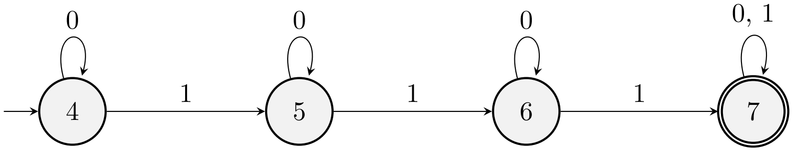
Theory of Computation: Assignment 3 Solutions

Arjun Chandrasekhar

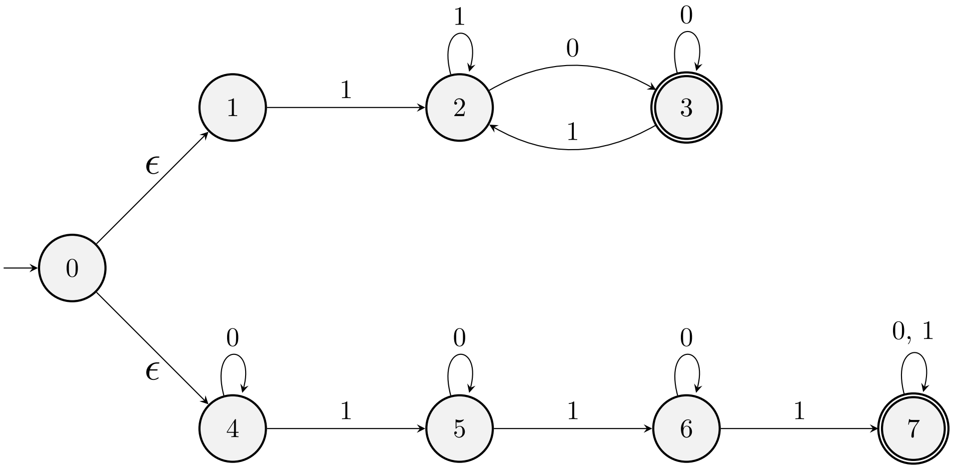
1. The following 3-State NFA recognizes A



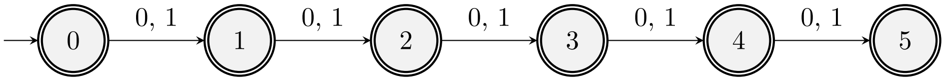
The following 4-state NFA recognizes B



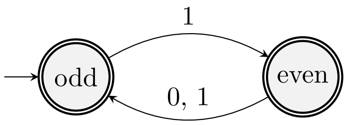
The following 8-state NFA recognizes $A \cup B$



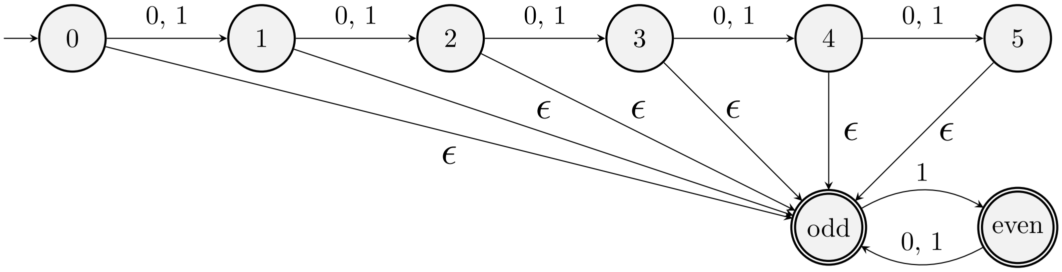
2. The following 6-state NFA recognizes A



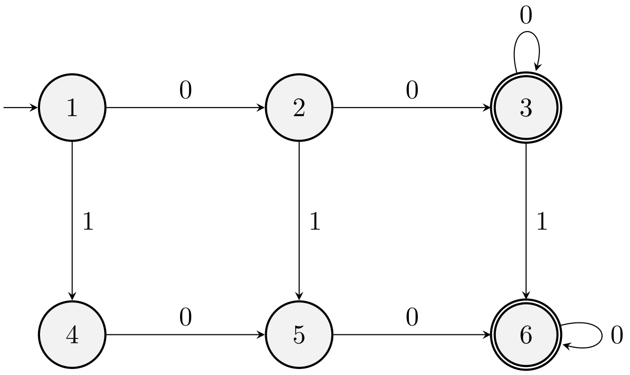
The following 2-state NFA recognizes B



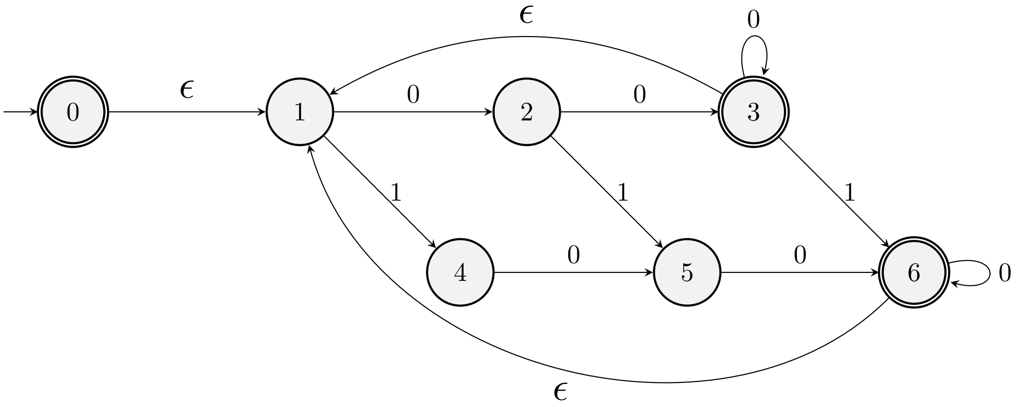
The following 8-state NFA recognizes $A \circ B$



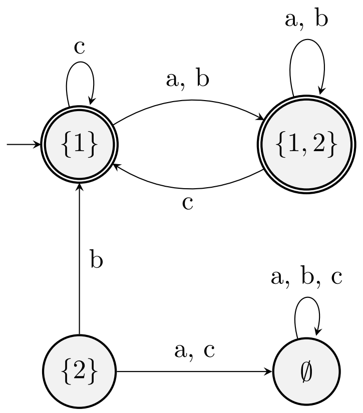
3. The following 6-state NFA recognizes A



The following 7-state NFA recognizes A^*



4. The following DFA is equivalent to the original NFA



Note that $\emptyset$ and $\{2\}$ are unreachable from the start state; they could be removed without affecting the language of the DFA.

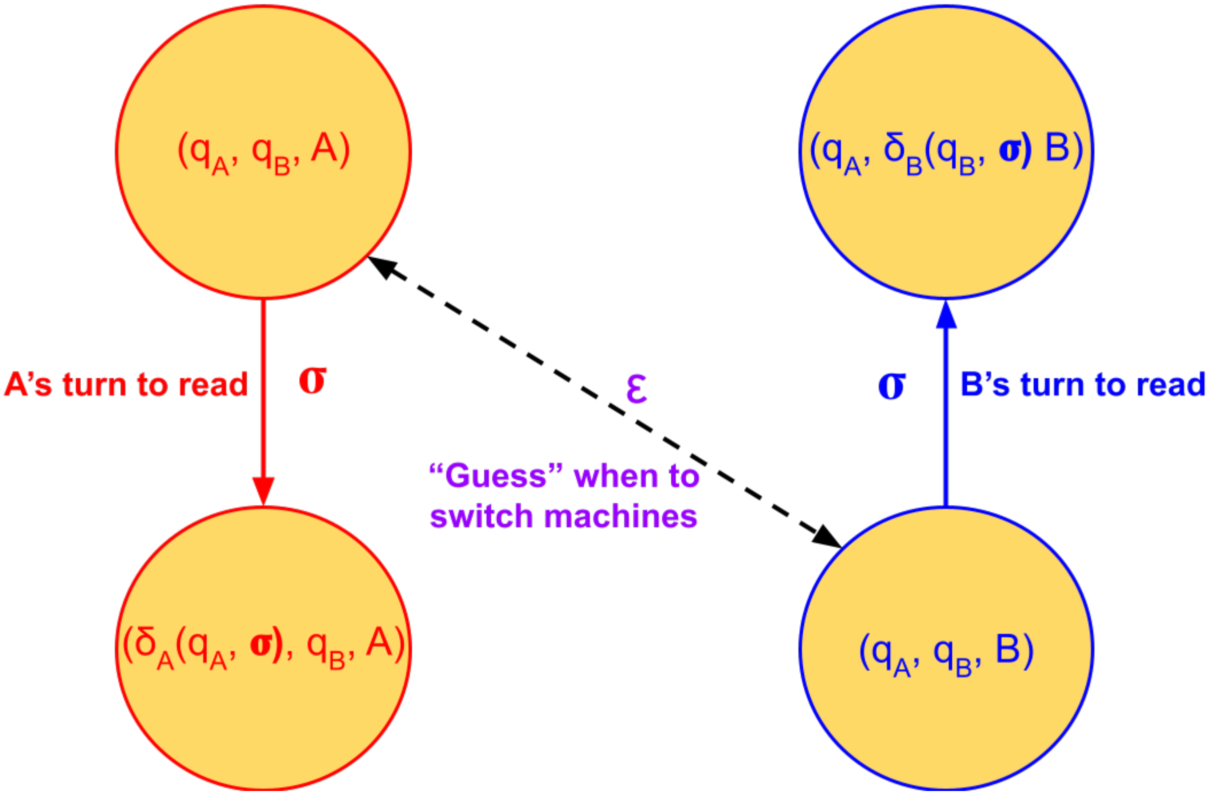
5. Following the hint, we will construct an NFA that runs A and B in alternation. However, we will not force the NFA to alternate after every character. Instead, it will stay with the same machine after it reads a character, and it will use ϵ transitions to "guess" when to switch machines.

   Formally, since A and B are regular, we know they are recognized by DFAs. Let $M_A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ be a DFA that recognizes A, and let $M_B = (Q_B, \Sigma, \delta_B, S_B, F_B)$ recognize B. To show that SHUFFLE(A, B) is regular, we will construct a DFA $M = (Q, \Sigma, \delta, S, F)$ it as follows:

   - $Q = Q_A \times Q_B \times \{A, B\}$ - each state is a combination of 1. a state from M_A 2. a state from M_B and 3. a "counter" that tells us if it's A's turn or B's turn to read a character
   - $\Sigma = \Sigma$ - the alphabet statys the same
   - δ is defined as follows:
     - $\delta((q_A, q_B, A), \sigma) = \{(\delta_A(q_A, \sigma), q_B, A)\}$ - If it's A's turn to read, we transition A's state while keeping B's state the same. It will stay A's turn until the machine "guesses" when to switch to B
     - $\delta((q_A, q_B, B), \sigma) = \{(q_A, \delta_B(q_B, \sigma), B)\}$ - If it's B's turn to read, we transition B's state while keeping A's state the same. It will stay B's turn until the machine "guesses" when to switch to A
     - $\delta((q_A, q_B, A), \epsilon) = \{(q_A, q_B, B)\}$ - at any point, we are allowed to "guess" that it's time to switch from A's turn to B's turn
     - $\delta((q_A, q_B, B), \epsilon) = \{(q_A, q_B, A)\}$ - at any point we are allowed to "guess" that it's time to switch from B's turn to A's turn
   - $S = (S_A, S_B, A)$ - at the start, we have A and B start in their respective start states. Nominally, we set it to A's turn at the start (although it doesn't matter because we can ϵ transition to B's turn before we read the first character).
   - $F = F_A \times F_B \times \{A, B\}$ - we accept any string that takes both A and B to one of their respective accept states. It can be either A's turn or B's turn at the end.

   Note that for NFAs, the output of the transition function is a *set of states*, rather than a single state.

   The following diagram illustrates the idea behind the construction.



6. (a) We will use a similar approach to the one we used to prove that NFAs recognize the regular languages. In particular, we will prove that every DFA has an equivalent all-NFA, and vice-versa.

($\Rightarrow$) First we will prove that if L is regular, then L is recognized by an all-NFA. We know that if L is regular, then it is recognized by a DFA D. As it turns out, D *is* an all-NFA that simply chooses not to use any non-determinism. Since a DFA only has one computation path, it technically accepts a string if and only if all paths accept. Thus, L is recognized by an all-NFA.

($\Leftarrow$) Next, we will prove that. if L is recognized by an all-NFA, it is regular. Let $M_A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ be the all-NFA that recognizes L. We will construct the following DFA $M = (Q, \Sigma, \delta, S, F)$ to recognize L:

- $Q = \mathcal{P}(Q_A)$ - every DFA state is a unique combination of states from M_A, representing where the all-NFA could potentially be at in its computation
- $\Sigma = \Sigma$ - the alphabet is the same
- $\delta(R \in Q, \sigma) = E\left(\bigcup_{r \in R} \delta_A(r, \sigma)\right)$ - let's break this down part by part
  - The input to the transition function is a DFA state R (which is really a combination of NFA states) and a symbol σ
  - We loop through all states $r \in R$, apply the all-NFA's transition function to each (r, σ), and combine (union) all the results
  - Finally, we take the epsilon closure of the states from the previous step; this reflects the fact that the all-NFA may take several ϵ transitions after reading σ
- $Q = E(\{Q_A\})$ - the start state is the combination of states that the all-NFA could be in before reading any symbols. This includes the nominal start state Q_A, as well as any states that can be reached from Q_A using ϵ transitions

- We will accept if we end up in a combination of states that are all accept states of the all-NFA. This means that every possible path led to an accept state. There are a few ways to write this:
  - $F = \{R | R \subseteq Q_A, R$ is non-empty and contains only accept states of $M_A\}$
  - $F = \{R | R \neq \emptyset, R \subseteq F_A\}$
  - $F = \mathcal{P}(F_A) \backslash \emptyset$ - the power set of the original accept states (with the empty set discarded)

(b) Because A and B are regular, there exist DFA's D_A and D_B that recognize A and B, respectively. To show that $A \cap B$ is regular, we will construct an all-NFA that recognizes $A \cap B$. Our all-NFA will contain all of the states and transitions of D_A and D_B. Additionally, we will have a special start state q_0 with ϵ transitions to the original start states of D_A and D_B. This way, the all-NFA can run the string through D_A or through D_B. However, because it is an all-NFA, it only accepts if *both* of these paths lead to accepting the string; thus, it only accepts the string if both D_A and D_B accept, which by definition gives us $A \cap B$.

7. First, we note that because Lucy can rotate the cups, and because Charlie can't see or feel the cups, there are really only 4 unique configurations:

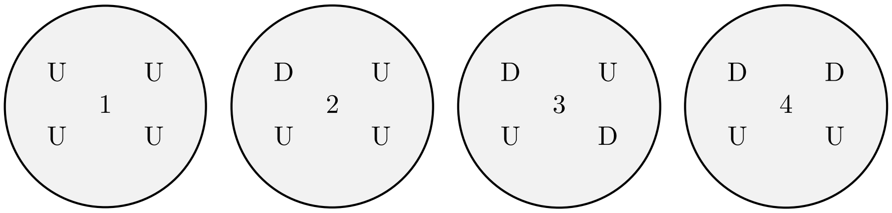
(a) All cups face the same way

(b) The diagonal cups match each other

(c) The left and right side cups match each other

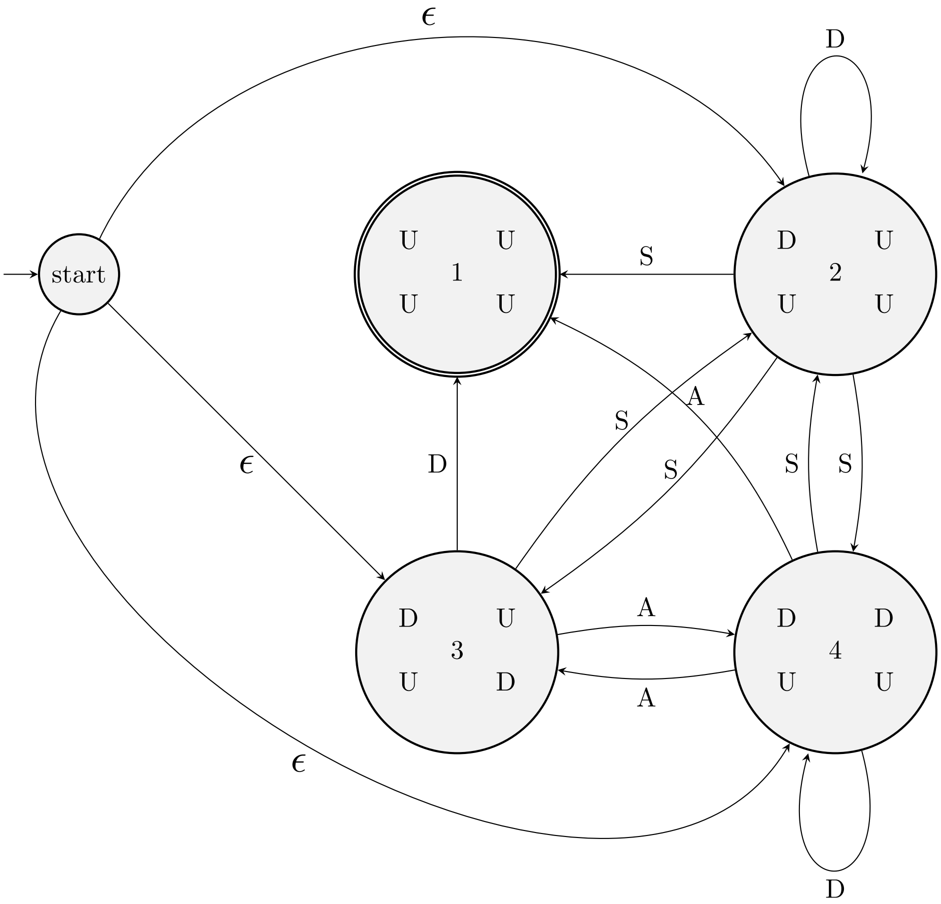
(d) Three cups match each other, with one odd cup out

Here are what those four configurations look like (D represents "down", and U represents "up").



Any other configuration is equivalent to one of the four above by switching up/down, and/or by rotating the tray.

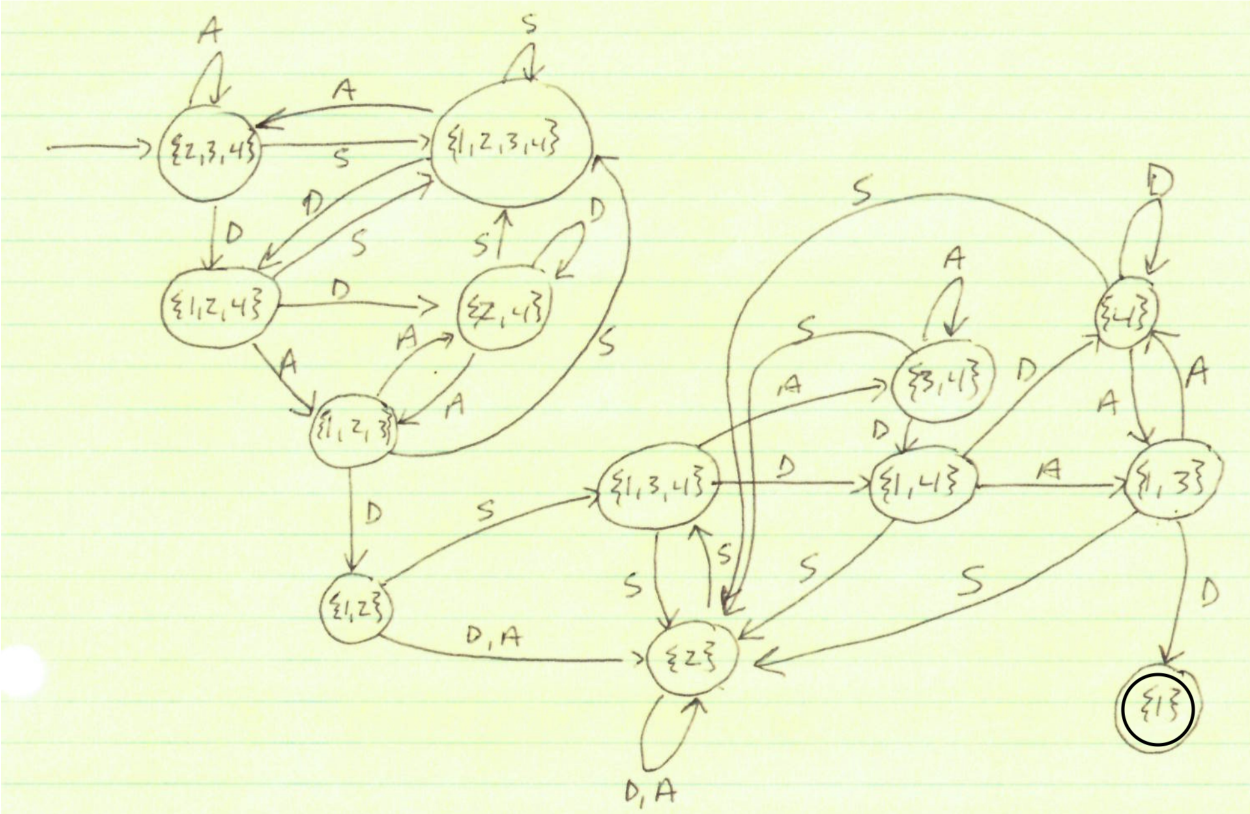
Now let's make an all-NFA to represent the moves that Charlie can make, and how they affect the configuration of the game. We'll use A to represent flipping adjacent cups, D to represent flipping diagonal cups, and S to represent flipping a single cup.



Some notes:

- Once again, remember that every possible configuration is equivalent to one of the four shown above through rotation and inverting up/down. So if you are in configuration 2, and you flip adjacent cups, you may not get the *exact* configuration in state 3 – but you will get something equivalent. If you are in state 4 and you flip adjacent cups, you might pick the right pair and win the game; or, you might flip the wrong pair, which leads you to a configuration that is equivalent (if not identical) to the one in state 3. If you are in state 3 and you flip diagonal cups, you are guaranteed to win! It doesn't matter what the exact state of the cups are - if the diagonals match, and you flip diagonal cups, then all the cups have to be facing the same way (which is either identical or equivalent to state 1).
- There is a special start state that has ϵ transitions to each of the non-winning configurations. This represents the fact that Snoopy can choose how to start the game, but he definitely won't choose to start the game with all cups facing the same way.
- We made this an all-NFA because we want to *guaranteed* that Charlie will win. We want a sequence of moves such that no matter how Snoopy rotates the tray, Charlie will get to state 1.

Then, we convert the all-NFA to a DFA. Here is the state diagram for the equivalent DFA:



Note that while many states include 1 as part of the combination, that's not good enough to be an accept state. That's because we need to be *guaranteed* to win. For example, the state $\{1, 2, 4\}$ is not an accept state; even though we could be in state 1, we could also potentially be in states 2 or 4, so any sequence that leads Charlie to this combination is not *guaranteed* to win. The only accept state is $\{1\}$, because it means that Charlie is guaranteed to win - he cannot possibly be in a losing configuration (assuming the game even made it this far and wasn't won earlier).

From the state diagram, we see that $\boxed{\textbf{DADSDAD}}$ is the shortest sequence that guarantees a win for Charlie. You can now use this to impress (and perhaps win a bet with) your friends who didn't have the foresight to take theory of computation with Dr. Chandrasekhar.