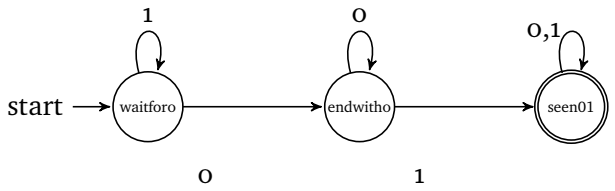
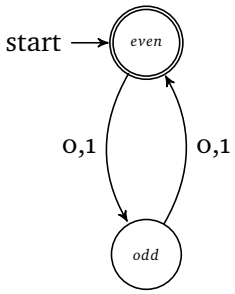
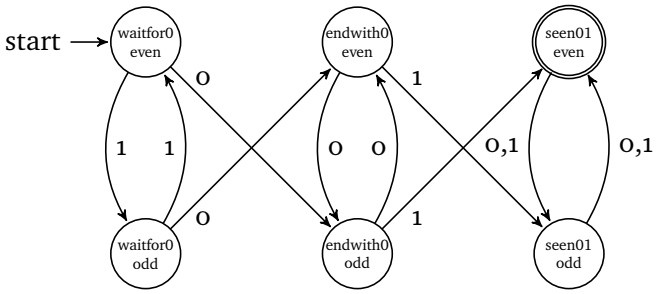
1. **Solution:** The language L can be seen to the intersection of two languages L_1 and L_2 where L_1 is the set of strings that contain 01 as a substring and L_2 is the set of strings of even length. We construct DFAs for L_1 and L_2 separately and then do a product construction to obtain a DFA for L.

   The two DFAs for L_1 and L_2 are shown below.





   The DFA after the product construction is shown below.



   The meaning of the states is given below.

(waitfor0,even): String does not end with a 0, does not contain 01, even length word.
(endwith0,even): String ends with a 0, does not contain 01, even length word.
(seen01,even): String contains 01, even length word.
(waitfor0,odd): String does not end with a 0, does not contain 01, odd length word.
(endwith0,odd): String ends with a 0, does not contain 01, odd length word.
(seen01,odd): String contains 01, odd length word.

■

2. **Solution:** $(0 + 1(11)^*0)^*(11)^*(0 + 1(11)^*0)^*(\epsilon + 1(11)^*)$

   We explain the above regular expression.

   The expression $(11)^*$ generates even length strings of 1s, and $1(11)^*$ generates odd length strings of 1s.

Strings with no even length blocks of 1s can then be represented as $(0 + 1(11)^*0)^*(\epsilon + 1(11)^*)$. The 0 in $1(11)^*0$ is used to separate each odd length block of 1s. The string can start with 0 and have different number of 0s between 1s, so we union 0 with $1(11)^*0$ in $(0 + 1(11)^*0)$. Since $(0 + 1(11)^*0)^*$ can only end with 0, we append $(\epsilon + 1(11)^*)$ so that the string can end with an odd length block of 1s.

Now we consider allowing one even length block of 1s. $(11)^*$ can be zero or one block of even number of 1s. A string with exactly one even length block of 1s can be split into a prefix and suffix with no even length of block of 1s with an even length length block of 1s in between. We thus consider the regular expression obtained by inserting $(11)^*$ between no blocks of 1s with even length to get the result $(0 + 1(11)^*0)^*(\epsilon + 1(11)^*)(11)^*(0 + 1(11)^*0)^*(\epsilon + 1(11)^*)$. The second $(0 + 1(11)^*0)^*$ can either start with 0 or not. If it is 0, it means to separate the block of 1s of even length with odd length. If it starts with 1, it concatenates an even block with an odd block, which makes it an odd block again, meaning our string do not have block of 1s of even length.

Also, we can see that the first $(\epsilon + 1(11)^*)$ is not necessary, since without it, the string can still end with 1s, using the second $(\epsilon + 1(11)^*)$. So the final answer is $(0 + 1(11)^*0)^*(11)^*(0 + 1(11)^*0)^*(\epsilon + 1(11)^*)$. ■

3. **Solution:** The solution is as follows:

   (a) i. $\emptyset$
   ii. ϵ
   iii. ϵ + a
   (b) $r_1' + r_2'$
   (c) $r_1' + r_1 r_2'$
   (d) $r_1^* r_1'$

   ■

4. **Solution:** Define the infinite set $F := \{b^j \mid j \geq 0\}$. We will show that F is a fooling set for the language L in question. Let $x := b^i$ and $y := b^j$ be arbitrary distinct strings in F. Assume further, without loss of generality, that $i < j$. Define $z := c^j$. We claim that z distinguishes x and y with respect to L. We have $xz = b^i c^j$ and since $i < j$, $xz \in L$. But $yz = b^j c^j$ and $yz \notin L$. Therefore, any pair of distinct strings in F are distinguishable with respect to L. Hence, F is a fooling set for L and hence L is not regular since F is infinite. ■

5. **Solution:** We define a CFG $G = (V, T, P, S)$ with start symbol S, set of terminals $T = \{a, b, c\}$, set of nonterminals $V = \{S, A, B\}$, and the following productions P:

$$\begin{aligned} S &\to Ac && \{a^i b^j c^k \mid i + j < k\} \\ A &\to Ac \mid aAc \mid B && \{a^i b^j c^k \mid i + j \leq k\} \\ B &\to bBc \mid \varepsilon && \{b^j c^j \mid j \geq 0\} \end{aligned}$$

   The role of each nonterminal is to generate the set of strings given after its production rules.

Our logic to generate strings where the number of 'c's is more than the sum of 'a's and 'b's ($i + j < k$), is to allow appending any number of 'c's at the end without adding 'a's and 'b's, and add one 'c' for every new 'a' and 'b'.

- The production rules to allow adding 'c's at the end are $A \to Ac \mid \varepsilon$ (the ε is for terminating this process).
- Increasing the number of 'a's (and hence 'c's) by 1 is done by appending 'a' at the beginning and 'c' at the end of a string. The production rules for allowing this are $A \to aAc \mid \varepsilon$.
- To increase the number of 'b's, we first add all 'a's in the string via the nonterminal A, and transition to another nonterminal B. Here we follow a similar process for adding 'b's as was done for 'a's, by appending a 'b' at the beginning and a 'c' at the end, through the rules $A \to B$ and $B \to bBc \mid \varepsilon$.
- The rules for A and B are combined in one grammar by removing the now redundant rule $A \to \varepsilon$ (as $A \to B$ followed by $B \to \varepsilon$ achieves the same). These productions are shown as the last two lines of the grammar defined above.
- The productions for A and B together generate all strings where the sum of 'a's and 'b's is at most the number of 'c's. To enforce the number of 'c's to be strictly more than this sum, we add one 'c' to every string in the first step, then go to the productions for adding 'a's (by using the rules for A). The rule for adding one 'c' and going to A is $S \to Ac$. Adding this rule completes the description of the grammar.

■

6. **Solution:** Define CFG $G = (V, T, P, S)$ as follows:

$$V = V_1 \cup V_2 \cup V_3 \cup \{S, A\}$$
$$T = T$$
$$P = P_1 \cup P_2 \cup P_3 \cup \{S \to S_1 | S_2 A, \; A \to S_3 A | \varepsilon\}$$
$$S = S$$

In this construction the production rule $S \to S_1$ generates L_1, $A \to S_3 A | \varepsilon$ generates L_3^*, and $S \to S_2 A$ generates $L_2 L_3^*$

■

7. **Solution:**

(a) Note that $M = (Q, \{0, 1\}, \delta, s, A)$ is a DFA that accepts the language L. Informally, we will construct a DFA that accepts flip(L) by changing all 0-transitions to 1-transitions and changing all 1-transitions to 0-transitions in M. It is clear that such a machine would accept flip(L) as $w \in$ flip(L) if and only if flip(w) $\in L$. Therefore, on input w, we want our machine to check if flip(w) is in L or not, and since we already have a DFA that accepts L, we simply flip all the bits of input w and check if it is accepted by M.

The only change we have to make to our new machine is in the transition function. Formally, let $M' = (Q, \{0,1\}, \delta', s, A)$ be a DFA where

$$\forall q \in Q,\ \delta'(q,0) = \delta(q,1)$$
$$\forall q \in Q,\ \delta'(q,1) = \delta(q,0).$$

M' accepts the language flip(L) for the reasons stated above.

(b) Again, $M = (Q, \{0,1\}, \delta, s, A)$ is a DFA that accepts the language L. Informally, we will construct an NFA that accepts flipsuffix(L) by guessing where the flipped suffix begins. We will make two copies of M and at every state in the first copy, there will be an ϵ-transition to the corresponding state in the second copy. These ϵ-transitions correspond to guessing where the flipped suffix begins. In the second copy of M, we again change all 0-transitions to 1-transitions and change all 1-transitions to 0-transitions as in part (a). This modification to the second copy will allow us to simulate flipped suffixes.

Formally, let $M' = (Q', \{0,1\}, \delta', s', A)$ be an NFA where

$$\begin{aligned}
Q' &= Q \times \{\text{before}, \text{after}\} \\
s' &= (s, \text{before}) \\
A' &= A \times \{\text{after}\} \\
\delta'((q,\text{before}), a) &= \{(\delta(q,a), \text{before})\} && \forall q \in Q,\ \forall a \in \{0,1\} \\
\delta'((q,\text{before}), \varepsilon) &= \{(q, \text{after})\} && \forall q \in Q \\
\delta'((q,\text{after}), 0) &= \{(\delta(q,1), \text{after})\} && \forall q \in Q \\
\delta'((q,\text{after}), 1) &= \{(\delta(q,0), \text{after})\} && \forall q \in Q.
\end{aligned}$$

M' accepts the language flipsuffix(L). ■