Problem 1

Let alphabet Σ be \{0, 1\}.

1. Construct a nondeterministic automaton whose language is the set of all strings that contain 010 as a substring.

2. Determinize your solution from part 1.

Problem 2

1. Let \( \textbf{P}_1 \) be the following reactive module:

   state:
   - \( \text{loc} = \{\text{noacc}, \text{req}, \text{acc}\} \)
   - \( \text{flag}_X = \{\text{down}, \text{up}\} \)

   external:
   - \( \text{flag}_Y = \{\text{down}, \text{up}\} \)

   initial:
   - \( \text{loc} = \text{noacc}; \text{flag}_X = \text{down} \)

   update:
   - \( \text{loc} = \text{noacc} \&\& \text{flag}_Y = \text{down} \rightarrow \text{loc} = \text{req} \)
   - \( \text{loc} = \text{req} \rightarrow \text{flag}_X = \text{up}; \text{loc} = \text{acc} \)
   - \( \text{loc} = \text{acc} \rightarrow \text{flag}_X = \text{down}; \text{loc} = \text{noacc} \)

Consider the system \( \textbf{P}_1 || \textbf{P}_1[\text{flag}_X/\text{flag}_Y, \text{flag}_Y/\text{flag}_X] \). Does this system ensure the safety property that the two processes are never at the acc location at the same time?

2. Let \( \textbf{P}_2 \) be the following reactive module:

   state:
   - \( \text{loc} = \{\text{noacc}, \text{req}, \text{acc}\} \)
   - \( \text{flag}_X = \{\text{down}, \text{up}\} \)

   external:
   - \( \text{flag}_Y = \{\text{down}, \text{up}\} \)

   initial:
   - \( \text{loc} = \text{noacc}; \text{flag}_X = \text{down} \)

   update:
   - \( \text{loc} = \text{noacc} \rightarrow \text{flag}_X = \text{up}; \text{loc} = \text{req} \)
   - \( \text{loc} = \text{req} \&\& \text{flag}_Y = \text{down} \rightarrow \text{loc} = \text{acc} \)
   - \( \text{loc} = \text{acc} \rightarrow \text{flag}_X = \text{down}; \text{loc} = \text{noacc} \)
Consider the system \( P_2 || P_2[\text{flag}_X/\text{flag}_Y, \text{flag}_Y/\text{flag}_X] \). Does this system ensure the safety property “no deadlock” that requires that in every state at least one of the processes can make progress?

3. Design a system \( P_3 \) that corrects the problems of \( P_1 \) and \( P_2 \). Hint: you need to add just one more variable.

Does your system ensure the liveness property “bounded waiting” that requires that if a process is waiting for access (i.e. is in the location \textit{req}), the other process can get access at most a bounded number of times?