Consider the sequential transmission of packets between two nodes (S and R) in a data network. The transmission channel is noisy and thus packets can get lost. One class of protocols that control the transmission is the class of STOP and WAIT protocols:

1. S sends a new packet to R and starts a timer

2a. S receives acknowledgement from R, go back to 1.

2b. S times out, S resends packet to R and restarts the timer, redo 2.

The receiver R receives packages and sends acknowledgements. A problem is that R needs to know if a packet received is a new packet or a retransmission of a previous packet by node S. This is because the transmission delay is not known, and the acknowledgements can also get lost. A strategy to address this problem is to add one bit to each packet header, resulting in "0-packets" and "1-packets". The header bit is flipped by node S each time S sends a new packet. The acknowledgements also includes a header bit, with values corresponding to the packet we acknowledge. This strategy is denoted the Alternating Bit Protocol (ABP).

Your tasks are (it is actually easiest to start with 2):

1. Model a communication channel with two nodes on each end using automata. It is possible for the sender and receiver to send at the same time, so it is good to split up the channel into two parts. We do not have to consider whether sending from both ends results in higher likelihood for loss of data, just that it is possible.

2. Specify automata for the sender and receiver so that they follow the ABP, and that all packets are transmitted to the receiver. Here you should take into account that everything is not observable; you can only observe what happens in your own end of the communication channel.

3. Check that the overall system is nonblocking and controllable.

4. Build supervisors for each end of the communication channel, and test that they are nonblocking and controllable. The two supervisors should operate independently, and not rely on events takin place in the other end, which will make the supervisors observable.

Hints:

1. As usual, a quite high level of abstraction makes things easier.

2. The timeout events must be uncontrollable, otherwise you can avoid a deadlock (caused by a mistake that is easy to do) by disabling timeout!

3. The overall automaton can get a lot of states, my version has 78 states.