Assignment 5

1. Consider a model for alternating bit protocol which simulates lost messages.

Program:

**mtype** = { msg, ack };

**chan** to\_sndr = [2] **of** { **mtype**, **bit** };

**chan** to\_rcvr = [2] **of** { **mtype**, **bit** };

**active** **proctype** Sender ()

{   **bit** seq\_out=1, seq\_in;

    do

    ::  to\_rcvr!msg (seq\_out) ->

        do

        :: to\_sndr?ack (seq\_in) ->

            if

            :: seq\_in == seq\_out ->

                seq\_out = !seq\_out

            :: else -> skip

            fi;

            break

        :: to\_sndr?ack (seq\_in) ->

            skip        */\* Message loss \*/*

        od

    od

}

**active** **proctype** Receiver ()

{   **bit** seq\_in, lost;

    do

    :: to\_rcvr?msg (seq\_in) ->

        to\_sndr!ack (seq\_in)

    :: to\_rcvr?msg (lost) ->

        skip            */\* Message loss \*/*

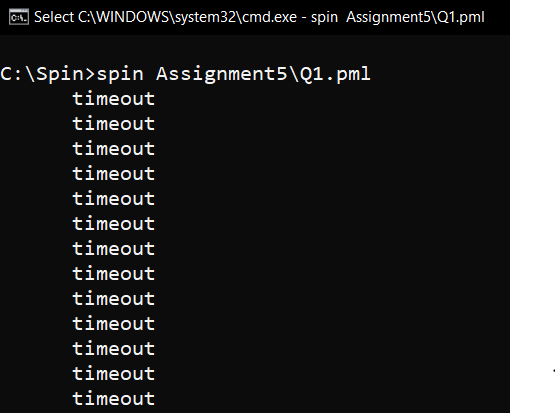
    :: timeout ->           */\* Recover from message loss \*/*

        to\_sndr!ack (seq\_in)

    od

}

Output:



1. Write a Promela model for Go-Back-N Sliding Window Protocol.

Program:

#define MaxSeq  3   */\* window size \*/*

#define Wrong(x)    x = (x+1) % (MaxSeq)

#define Right(x)    x = (x+1) % (MaxSeq + 1)

#define inc(x)      Right(x)

*/\* file ex.9 \*/*

**chan** q[2] = [MaxSeq] **of** { **byte**, **byte** }; */\* message passing channel \*/*

**active** [2] **proctype** p5()    */\* starts two copies of proctype p5 \*/*

{   **byte**    NextFrame, AckExp, FrameExp, r, s, nbuf, i;

**chan**    in, out;

    in  = q[\_pid];

    out = q[1-\_pid];

    xr in; xs out;      */\* partial order reduction claims \*/*

    do

    :: nbuf < MaxSeq -> */\* outgoing messages \*/*

        nbuf++;

        out!NextFrame , (FrameExp + MaxSeq) % (MaxSeq + 1);

        inc(NextFrame)

    :: q[\_pid]?r,s ->   */\* incoming messages \*/*

        if

        :: r == FrameExp ->

            printf("MSC: accept %d\n", r);

            inc(FrameExp)

        :: else */\* ignore message \*/*

        fi;

        do

        :: ((AckExp <= s) && (s <  NextFrame)) || ((AckExp <= s) && (NextFrame <  AckExp)) || ((s <  NextFrame) && (NextFrame <  AckExp)) ->

            nbuf--;

            inc(AckExp)

        :: else -> break

        od

    :: timeout ->       */\* retransmission timeout \*/*

        NextFrame = AckExp;

        printf("MSC: timeout\n");

        i = 1;

        do

        :: i <= nbuf ->

            out!NextFrame , (FrameExp + MaxSeq) % (MaxSeq + 1);

            inc(NextFrame);

            i++

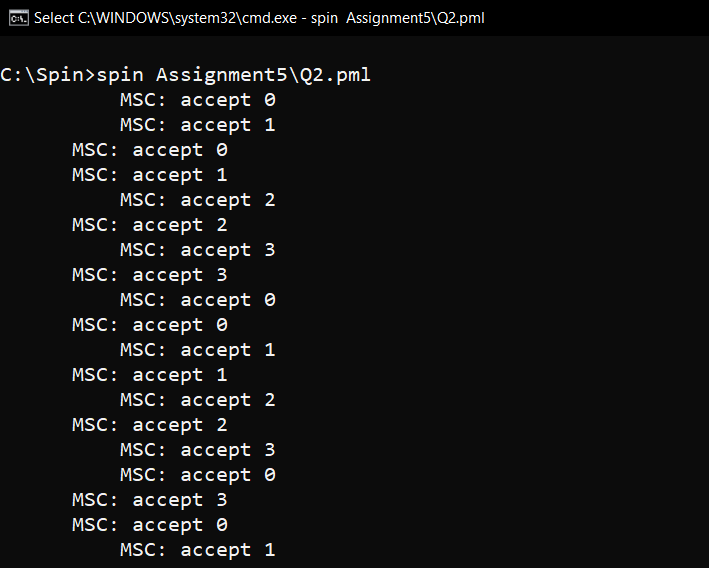
        :: else -> break

        od

    od

}

Output:



1. Consider the Petri Net shown below. Build a Spin model for this Petri Net and verify whether it is deadlock-free.

Program:

#define Place   **byte**

Place p1, p2, p3;

Place p4, p5, p6;

#define inp1(x)     (x>0) -> x=x-1

#define inp2(x,y)   (x>0&&y>0) -> x = x-1; y=y-1

#define out1(x)     x=x+1

#define out2(x,y)   x=x+1; y=y+1

**init**

{   p1 = 1; p4 = 1; */\* initial marking \*/*

    do

        :: atomic { inp1(p1)    -> out1(p2) }

        :: atomic { inp2(p2,p4) -> out1(p3) }

        :: atomic { inp1(p3)    -> out2(p1,p4) }

        :: atomic { inp1(p4)    -> out1(p5) }

        :: atomic { inp2(p1,p5) -> out1(p6) }

        :: atomic { inp1(p6)    -> out2(p4,p1) }

    od

}

Output:

