# Temporal Logic Specification and Analysis

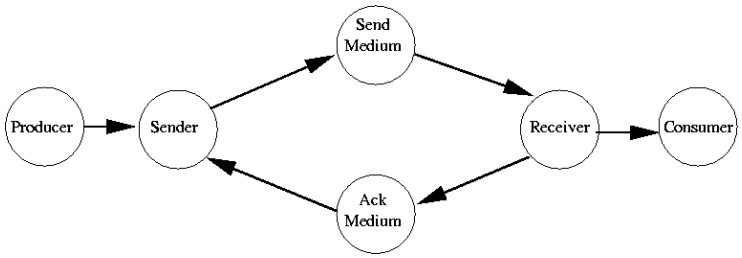
David Kendall

## 1 Objectives

- Make further progress with SPIN and PROMELA in modelling and analysing some simple concurrent systems
- Specify properties using temporal logic

## 2 The alternating bit protocol

The figure below shows the architecture of the Alternating Bit Protocol.




The producer wishes to send data to the consumer, reliably, through a medium which might be lossy or error-prone. The alternating bit protocol accomplishes this as follows:

Each data message sent by the sender contains a protocol bit, 0 or 1. When the sender sends a message, it sends it repeatedly (with its protocol bit) until receiving an acknowledgment (ACK) from the receiver that contains the same protocol bit as the message being sent. When the receiver receives a message(with error-free checksum, etc), it sends an ACK to the sender and includes the protocol bit of the message received. The first time the message is received, the protocol delivers the message to the consumer. Subsequent messages with the same bit are simply acknowledged. When the sender receives an acknowledgment containing the same bit as the message it is currently transmitting, it stops transmitting that message, flips the protocol bit, and repeats the protocol for the next message.

Does this protocol work? The following PROMELA code is a first cut at modelling the protocol as four processes: a sender, a receiver, a message send medium, and an acknowledgement medium. How will you check that it works?

It models error-free media. How will you introduce errors into the transmission media and check that it still works?

Specify some properties that you think the model should exhibit. Write the properties in English and in LTL. Use the SPIN verifier to check the LTL formulas.

**Simple model with a 'perfect' channel**

```
1  /* Basic version of ABP. No errors in the transmission media. */
2
3  #define INIT_BIT 0          /* Start with a zero-tagged message */
4  chan stran = [0] of {bit}; /* Sender -> Transmit Medium */
5  chan rtran = [0] of {bit}; /* Transmit Medium -> Receiver */
6  chan rack = [0] of {bit};  /* Receiver -> Ack medium */
7  chan sack = [0] of {bit};  /* Ack medium -> Sender */
8
9  bit input = 0;
10 bit deliver = 0;
11
12 /* Model the sending process */
13 proctype Send () {
14   bit stag = INIT_BIT; /* Tag on message to send */
15   bit atag;            /* Tag on acknowledgment */
16 do
17 :: input = 0;
18    do
19    :: break              /* got new input to send  */
20    :: true -> skip       /* just waiting for input */
21    od;
22    input = 1;
23    do
24    :: stran!stag;
25      if
26      :: sack?atag ->
27         if
28         :: (atag == stag) -> break
29         :: else -> skip
30         fi
31      :: timeout -> skip
32      fi
33    od;
34    stag = 1-stag
35 od
36 }
37 /* Model the transmission medium for messages
38 from sender to receiver */
```

```
39 proctype Trans () {
40 bit m;
41 do
42 :: stran?m;
43    rtran!m
44 od
45 }
46 /* Model the transmission medium for acknowledgments
47 from receiver to sender */
48 proctype Ack () {
49 bit a;
50 do
51 :: rack?a;
52    sack!a
53 od
54 }
55 /* Model the receiving process */
56 proctype Receive () {
57 bit etag = INIT_BIT; /* Expected tag on received message */
58 bit rtag;            /* Actual tag on received message */
59 do
60 :: deliver = 0;
61    do
62    :: rtran?rtag;
63       if
64       :: (rtag == etag) -> break
65       :: else -> rack!rtag
66       fi
67    od;
68    deliver = 1;  /* deliver message */
69    rack!rtag;
70    etag = 1-etag
71 od
72 }
73
74 init {
75   atomic {
76     run Send(); run Trans(); run Ack(); run Receive()
77   }
78 }
```