SPIN: Introduction and Examples *

Alessandra Giordani
agiordani@disi.unitn.it
http://disi.unitn.it/~agiordani

Formal Methods Lab Class, September 28, 2014

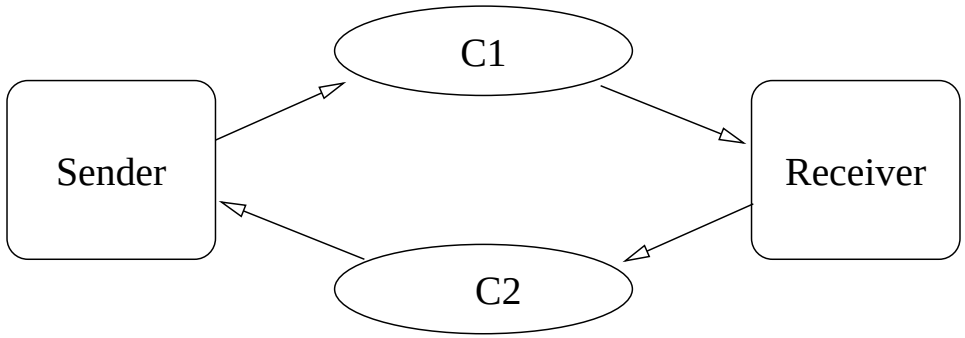
Università degli Studi di Trento

*These slides are derived from those by Stefano Tonetta, Alberto Griggio, Silvia Tomasi, Thi Thieu Hoa Le for FM lab 2011/13

Contents

Reliable FIFO Communication Problem

- We want to implement a reliable FIFO communication using less reliable channels.
- A user *Sender* sends messages to another user *Receiver* by means of two channels C_1 and C_2
- C_1 and C_2 are non-reliable channels.
- The non-reliable channels may lose or duplicate the messages.



Alternating Bit Protocol

- *Sender* tags the messages with an alternating bit (e.g. it sends `(msg1, 0)`, `(msg2, 1)`, `(msg3, 0)`, ...).
- *Sender* repeatedly sends a message with its tag until it receives a bit acknowledgment from *Receiver*.
- Suppose *Sender* has sent `(msg, tag)` and receives `b` as acknowledgment:
  - if `b` is equal to `tag`, then it means that *Receiver* has received the right message, so it sends a new message with a different value for `tag`;
  - otherwise it sends `(msg, tag)` again.
- Similarly, suppose *Receiver* receives `(msg, tag)`:
  - if `tag` is different from the last received bit, then it means that it is a new message;
  - otherwise, the message is old.

  In both cases, *Receiver* sends `tag` back to *Sender* to communicate the correct receipt of the message.

Alternating Bit Protocol

```
mtype = { msg, ack };

chan to_sndr = [2] of { mtype, bit };
chan to_rcvr = [2] of { mtype, bit , int };

active proctype Sender()
{
...
}

active proctype Receiver()
{
...
}
```

Alternating Bit Protocol

```
active proctype Sender()
{       bit seq_out, seq_in;
        int next_msg;
       do
       :: to_rcvr!msg(seq_out, next_msg) ->
              to_sndr?ack(seq_in);
              if
              :: seq_in == seq_out ->
                     /* obtain new message */
                     seq_out = 1 - seq_out;
                     next_msg++
              :: else
              fi
       od
}
```

Alternating Bit Protocol

```
active proctype Receiver()
{       bit seq_in, last_seq_in;
        int received;
        do
        :: to_rcvr?msg(seq_in, received) ->
           if
             :: (seq_in != last_seq_in) ->
                printf("Received: %d\n", received);
                last_seq_in = seq_in
             :: else
           fi
        :: to_sndr!ack(seq_in)
        od
}
```

Exercise 1

- Try with:

```
active proctype Receiver()
{       bit seq_in, last_seq_in;
        int received;
        do
        :: to_rcvr?msg(seq_in, received) ->
           if
             :: (seq_in != last_seq_in) ->
                printf("Received: %d\n", received);
                last_seq_in = seq_in
             :: else
           fi;
           to_sndr!ack(seq_in)
        od
}
```

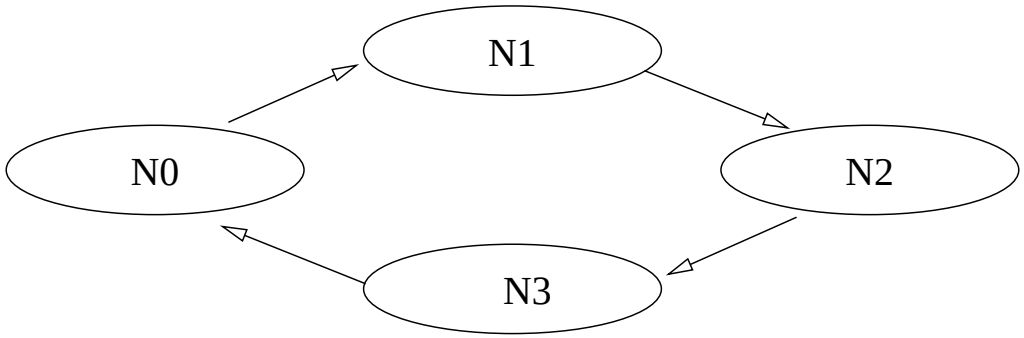
Exercise 2

- Model the non-reliability with:

```
inline risky_sending(channel, type, tag, val)
{
  if
    :: channel!type(tag, val)
    :: channel!type(tag, val); channel!type(tag, val);
    :: true
  fi
}
```

Leader Election Problem

- N processes are the nodes of a unidirectional ring network: each process can send messages to its clockwise neighbor and receive messages from its counterclockwise neighbor.
- The requirement is that, eventually, only one process will output that it is the leader.
- We assume that every process has a unique identifier.
- The leader must have the highest identifier



Exercise 3: Le Lann, Chang, Roberts (LCR) solution

- Initially, every process passes its identifier to its successor.
- When a process receives an identifier from its predecessor, then:
  - if it is greater than its own, it keeps passing the identifier;
  - if it is smaller, it discards the identifier;
  - if it is equal to its own identifier, it declares itself leader:
    - the leader informs the others that it is the leader;
    - after a process receives the message with the id of the leader, it exits.

Hint:

```
mtype = { candidate, leader };
chan c[N] = [BUFSIZE] of { mtype, byte };

proctype node(chan prev, next) { ... }

init {
   ...
   do
     :: proc <= N -> run node(...);
   ...
}
```