Networking model checking in Spin

Beerend Lauwers Frank Wijmans

April 3, 2012

Table of contents

Datatypes

We modelled the data as follows:

- Number of users: 4
- Number of channels: 1 to be filled with F_message
- F_message - a typedef containing a sender and constraint.

Datatypes

```
#define MAXUSERS 4
#define MAXMESSAGES 1

F_message recMsg;

typedef F_message
{
  byte sender;
  bit constraint;
}

chan recipients[MAXUSERS] = [MAXMESSAGES] of {
    F_message };
```

Introduction

- `receiveMsg( byte` *receiver* `);` This process type takes messages from a given channel (recipients[receiver])
- `sendMsg( byte` *sender*`, bit` *constraint* `);` Creates messages, and sends them only to the followers (or followers' followers) according to the constraint.
- `init();` Start to receive and send messages. Only 3 for every user. $Numberofprocesses = 3 * MAXUSER$

Receive messages

```
proctype receiveMsg( byte receiver )
{
F_message recMsg;
do
:: recipients[ receiver ] ? recMsg;
byte fol1,fol2;
fol1 = (recMsg.sender+1)%MAXUSERS;
fol2 = (recMsg.sender+2)%MAXUSERS;
assert ((fol1 == receiver) || (fol2 == receiver)
|| ((fol1+2)%MAXUSERS == receiver) || ((fol1+2)%
    MAXUSERS == receiver)
|| ((fol2+2)%MAXUSERS == receiver) || ((fol2+2)%
    MAXUSERS == receiver)
);
od;
}
```

Sending messages

```
proctype sendMsg ( byte sender; byte constraint )
{
F_message newMsg;
byte fol1, fol2, fol11, fol12, fol21, fol22;

\*
 * Locally save followers id's
 *\

d_step{
newMsg.sender = sender;
newMsg.constraint = constraint;

fol1 = ((sender+1)%MAXUSERS);
fol2 = ((sender+2)%MAXUSERS);
fol11 = ((fol1+1)%MAXUSERS);
fol12 = ((fol1+2)%MAXUSERS);
fol21 = ((fol2+1)%MAXUSERS);
fol22 = ((fol2+2)%MAXUSERS);
```

Sending messages (2)

```
}
do
::   if
:: (sender > 0) -> atomic{ /*Send to Fo*/
if
:: fol1 != sender -> recipients[fol1] ! newMsg
:: else skip
fi;
if
:: fol2 != sender -> recipients[fol2] ! newMsg
:: else skip
fi;
}
   if
```

Sending messages (3)

```
:: (constraint == 1) -> atomic{ /*Send to Fo2*/
   if
   :: sender != fol11 -> recipients[fol11] ! newMsg
   :: else skip
   fi;
   if
   :: sender != fol12 -> recipients[fol12] ! newMsg
   :: else skip
   fi;
if
:: sender != fol21 -> recipients[fol21] ! newMsg
:: else skip
fi;
```

Sending messages (4)

```
if
:: sender != fol22 -> recipients[fol22] ! newMsg
:: else skip
fi
}
:: else skip
fi
:: else skip
fi
od;
}
```

Initializing

```
init
{
/*
* Start all receivers and senders, for every
    channel (maxusers)
*/

byte i = 0;

atomic{
do
:: (i < MAXUSERS) -> run receiveMsg(i); run sendMsg
    (i,0); run sendMsg (i,1); i++;
:: else -> break;
od;
}
}
```

Asserts

- Privacy constraint is never broken
- No deadlocks
- Message eventually reaches the receiver

Work in progress

- How to check whether all messages eventually reach the right user.
  - Implicitly done, it would deadlock otherwise, but is this enough?
  - Checking on the right follower relation in every message.
- How to express assertions in LTL.
  - Using never claim