SPIN assignment – report

Krzysztof Pszeniczny

17th April 2016

1 Implementation of mailboxes

I have implemented mailboxes as ordinary promela channels, setting 255 as their length limit. It can be easily seen that in the test programs at most n^2 messages may be present at once in any channel, where n denotes the number of processes.

As the number of states grows exponentially in the number of processes, verification of the required properties is not feasible for $n > 16$ and would stay so even if the channels were unbounded.

1.1 Simplifying assumptions

In the test program, the sender sends its own identifier as the message content and it sends just one message to each of the processes. Hence, no message id checks were necessary – every message is uniquely determined by its sender and receiver (and the message content).

Moreover, the processes are indistinguishable, so it is sufficient to check all the properties only for two cases:

- Process 1 sends a message with contents 1 to itself
- Process 1 sends a message with contents 1 to process 2

1.1.1 Auxiliary definitions

I have introduced auxiliary definitions:

$$\begin{aligned} \texttt{send}(x, y, msg) &:= x_{\texttt{proc\_send}} \wedge sm_x = msg \wedge tp_x = y \\ \texttt{receive}(x, msg) &:= x_{\texttt{proc\_receive}} \wedge rm_x = msg \\ \texttt{receives\_infinitely\_often}(to) &:= \Box\Diamond \texttt{proc\_receive}_{to} \end{aligned}$$

Hence `send`(x, y, msg) means that process x is about to send message msg to process y, `receive`(x, msg) means that process x has just received a message msg.

1.2 `mb_guaranteed_delivery`

As noted above – by symmetry it is sufficient to verify the property only when 1 is the sender and either 1 or 2 is the receiver. Hence, I introduced an auxiliary definition `guaranteed_delivery`$(from, to, msg)$ which is a direct translation of the formula from the problem statement, i.e.:

$$\Box\,(\texttt{send}(from, to, msg) \wedge \texttt{receives\_infinitely\_often}(to) \implies \texttt{receive}(to, msg))$$

1.3 `mb_at_most_once_delivery`

Once again, I have introduced an auxiliary macro `at_most_once_delivery`(to, msg) defined as

$$\Box\,(\texttt{receive}(to, msg) \implies \Box\,(\texttt{end\_proc\_await}_{to} \implies \neg\Diamond\texttt{receive}(to, msg)))$$

This formula means that if to receives a message msg then it will never again receive the same message after it makes some progress: the process may stay idle for a while, preempted by other processes and `receive`(to, msg) will stay true until the process performs `rm = 0`. Hence, it was not possible to use the X operator to formalise this property – moreover, using X may cause the LTL formula to stop being stutter-invariant, which may yield incorrect results when partial order reduction is on.[1]

1.4 `mb_no_fabrication`

As above, I introduced an auxiliary macro `mb_no_fabrication`$(from, to, msg)$ checking that a message msg is not received by to until it is sent by $from$:

$$\neg\texttt{receive}(to, msg)\,W\,\texttt{send}(from, to, msg)$$

The 'weak until' operator W is used, because I also account for the possibility that $from$ may refrain from sending msg to to.

[1] However, I had to switch partial order reduction off, because my formalisation accesses local variables of processes ('remote varrefs', in spin's parlance).

2 Broadcast algorithms

2.1 Summary

property	basic broadcast	reliable broadcast
`no_duplication`	+	+
`no_creation`	+	+
`weak_validity`	+	+
`strong_validity`	+	+
`weak_agreement`	−	+
`strong_agreement`	−	−

State explosion necessitated using `-DONE_BROADCASTER` when verifying the reliable broadcast.

2.2 Formalisation of the properties

This time, the processes are no longer indistinguishable, because `bb_broadcast` sends messages in a particular order. Hence it is not possible for a process to send a message to 3 but not to 1, while it is perfectly legal for it to send a message to 1 and then fail.

This necessitated verification for all triples (`receiver`, `sender`, `third`) of processes. The corresponding process identifiers are defined as macros by my script running spin, which performs the verification for each of the 27 possible triples (I set the number of processes to 3).

Fortunately, message contents are unique so I could use them to compare messages.

2.2.1 `no_duplication`

This property has been formalised entirely analogously to `mb_no_duplication`, with the same caveats causing me not to use the X operator.

2.2.2 `no_creation`

This property has been formalised entirely analogously to `mb_no_creation`, with the same caveats causing me to use the W operator.

2.2.3 `weak_validity`

I have defined an auxiliary macro `weak_validity`(who, msg) which asserts that if who is a correct process then every time it broadcasts the message, it

will also deliver it later:

$$\texttt{correct}(who) \implies \texttt{if\_broadcast\_then\_delivered}(who, who, msg)$$

It uses auxiliary macros: $\texttt{correct}(who)$ defined as:

$$\Box\neg\texttt{end\_proc\_fail}_{who}$$

and $\texttt{if\_broadcast\_then\_delivered}(who, sender, msg)$ defined as

$$\Box(\texttt{broadcast}(sender, msg) \implies \Diamond\texttt{delivered}(who, msg))$$

where `broadcast` and `delivered` are analogous to `send` and `receive` used to verify mailboxes.

2.2.4 `strong_validity`

This property is analogous to the previous one:

$$\texttt{correct}(sender)\wedge\texttt{correct}(who) \implies \texttt{if\_broadcast\_then\_delivered}(who, sender, msg)$$

2.2.5 `weak_agreement`

I have defined an auxiliary macro $\texttt{if\_delivered\_then\_delivered}(who, first, msg)$ asserting that if $first$ eventually receives the message msg, then the same can be said of who:

$$\Diamond\texttt{delivered}(first, msg) \implies \Diamond\texttt{delivered}(who, msg)$$

With this definition in hand, we may define $\texttt{weak\_agreement}(who, msg)$ as:

$$\texttt{correct}(who)\wedge\texttt{correct}(other) \implies \texttt{if\_delivered\_then\_delivered}(who, other, msg)$$

2.2.6 `strong_agreement`

This property is formalised as:

$$\texttt{correct}(who) \implies \texttt{if\_delivered\_then\_delivered}(who, other, msg)$$

2.3 Trails

2.3.1 `bb_weak_agreement`

The counterexample found by spin used options: `-DNUM_PROCESSES=3 -DSENDER=1 -DRECEIVER=3 -DTHIRD=2`.

In this counterexample process 1 (the sender) failed between sending the message to the processes 2 (third) and 3 (receiver). Hence, although the third process (i.e. 2) has received the message and both the receiver and the third process are correct, the receiver will never receive the message.

2.3.2 `bb_strong_agreement`

The same counterexample was found for `bb_strong_agreement`.

2.3.3 `rb_strong_agreement`

The options that led spin to a counterexample were: `-DNUM_PROCESSES=3 -DRELIABLE_BROADCAST -DSENDER=1 -DRECEIVER=3 -DTHIRD=2 -DONE_BROADCASTER=1`.

Process 1 failed after sending its message to 2, but before sending it to 3. Process 2 then delivered the message and, in the `rb_continuation` phase, tried to send it to every process, but a failure occurred just before sending it to 3.

Hence, although 2 delivered the message, a correct process 3 will never deliver it.