SPIN Assignment
CS4211 AY20/21 Semester 1

Au Liang Jun (A0173593W)
e0203163@u.nus.edu

1. Communication between Stations

1.1 LTL Property

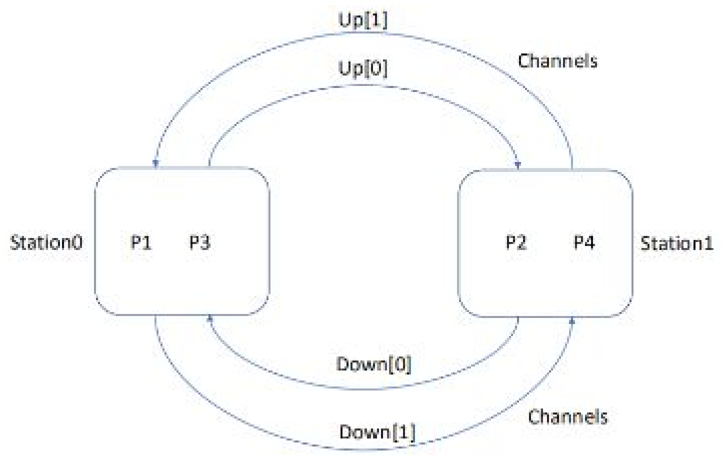
The LTL property used is as follows:

```
#define p1 []((down[1]?<start>) -> <>(up[1]?<stop>))
#define p2 []((down[0]?<start>) -> <>(up[0]?<stop>))
#define p3 []((up[0]?<start>) -> <>(down[0]?<stop>))
#define p4 []((up[1]?<start>) -> <>(down[1]?<stop>))
            ltl q {p1 && p2 && p3 && p4}
```

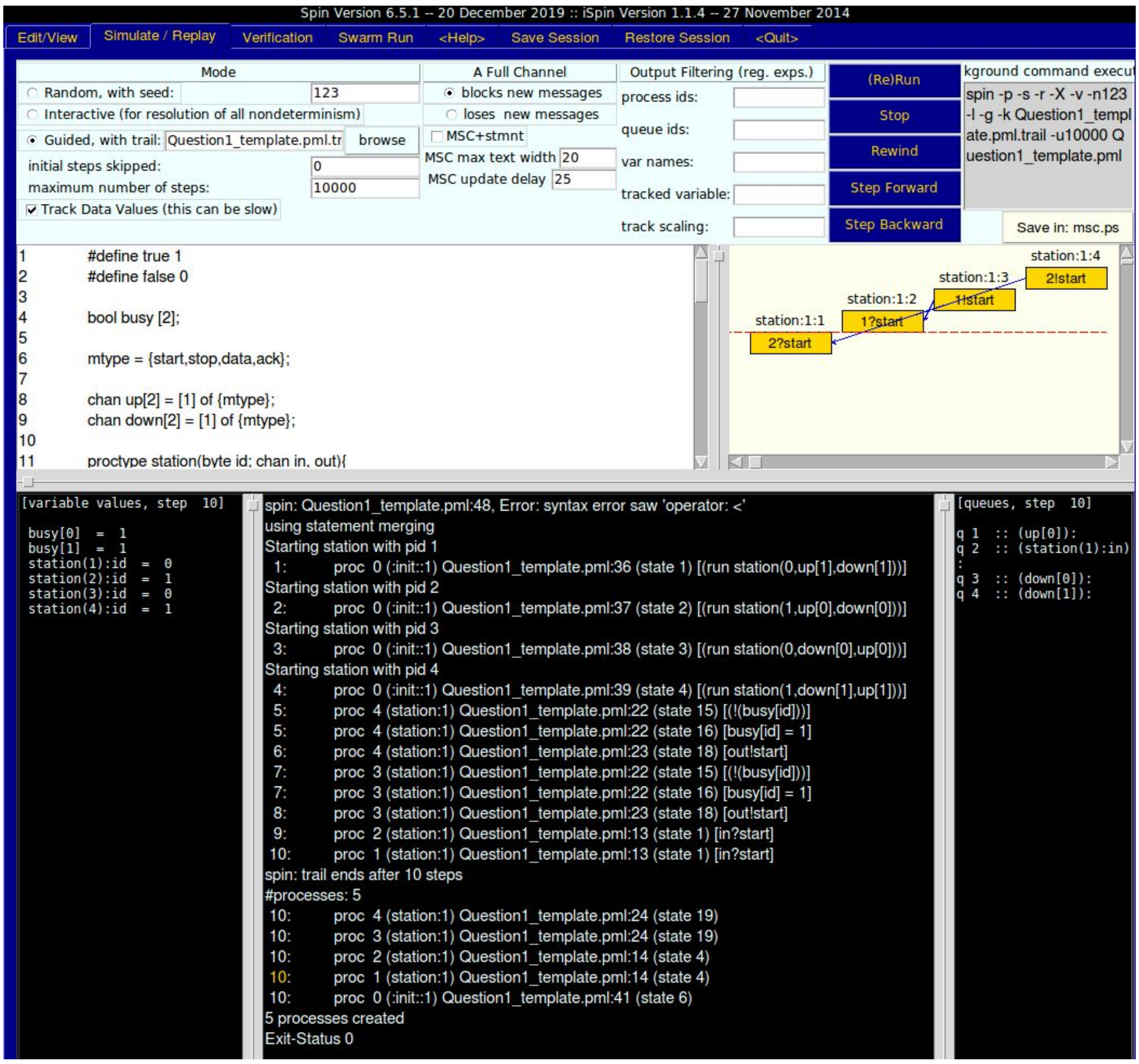
The LTL property is expressed with the help of the angled bracket channel receive operation. This checks for the existence of an item in a buffer, without removing it from the buffer. In words, the LTL I've written checks that for each station process, if a start message is sent into the out channel, a stop message is eventually received in the in channel. This signifies that a communication that is started will always end.

The LTL can be found in Question1_template.pml and Question1.pml.

1.2. Model Augmentation



The property is untrue as the protocol deadlocks in certain execution traces. It can be observed that when an operator, say P1 from the diagram above, begins sending data, it is possible for operators from the opposite station (P2 and P4 in our example) to also begin sending data. With both stations busy sending data, neither are free to receive data, resulting in a deadlock. The LTL property is thus violated.



Mode
- Random, with seed: 123
- Interactive (for resolution of all nondeterminism)
- [✓ Guided, with trail: Question1_template.pml.tr] browse

initial steps skipped: 0
maximum number of steps: 10000
[✓ Track Data Values (this can be slow)]

A Full Channel
- [✓ blocks new messages]
- loses new messages

MSC+stmnt
MSC max text width 20
MSC update delay 25

Output Filtering (reg. exps.)
process ids:
queue ids:
var names:
tracked variable:
track scaling:

kground command execu
```
spin -p -s -r -X -v -n123
-l -g -k Question1_templ
ate.pml.trail -u10000 Q
uestion1_template.pml
```
Save in: msc.ps

```
#define true 1
#define false 0

bool busy [2];

mtype = {start,stop,data,ack};

chan up[2] = [1] of {mtype};
chan down[2] = [1] of {mtype};

proctype station(byte id; chan in, out){
```

```
[variable values, step  10]

 busy[0]   =  1
 busy[1]   =  1
 station(1):id  =  0
 station(2):id  =  1
 station(3):id  =  0
 station(4):id  =  1
```

```
spin: Question1_template.pml:48, Error: syntax error saw 'operator: <'
using statement merging
Starting station with pid 1
  1:    proc  0 (:init::1) Question1_template.pml:36 (state 1) [(run station(0,up[1],down[1]))]
Starting station with pid 2
  2:    proc  0 (:init::1) Question1_template.pml:37 (state 2) [(run station(1,up[0],down[0]))]
Starting station with pid 3
  3:    proc  0 (:init::1) Question1_template.pml:38 (state 3) [(run station(0,down[0],up[0]))]
Starting station with pid 4
  4:    proc  0 (:init::1) Question1_template.pml:39 (state 4) [(run station(1,down[1],up[1]))]
  5:    proc  4 (station:1) Question1_template.pml:22 (state 15) [(!(busy[id]))]
  5:    proc  4 (station:1) Question1_template.pml:22 (state 16) [busy[id] = 1]
  6:    proc  4 (station:1) Question1_template.pml:23 (state 18) [out!start]
  7:    proc  3 (station:1) Question1_template.pml:22 (state 15) [(!(busy[id]))]
  7:    proc  3 (station:1) Question1_template.pml:22 (state 16) [busy[id] = 1]
  8:    proc  3 (station:1) Question1_template.pml:23 (state 18) [out!start]
  9:    proc  2 (station:1) Question1_template.pml:13 (state 1) [in?start]
 10:    proc  1 (station:1) Question1_template.pml:13 (state 1) [in?start]
spin: trail ends after 10 steps
#processes: 5
 10:    proc  4 (station:1) Question1_template.pml:24 (state 19)
 10:    proc  3 (station:1) Question1_template.pml:24 (state 19)
 10:    proc  2 (station:1) Question1_template.pml:14 (state 4)
 10:    proc  1 (station:1) Question1_template.pml:14 (state 4)
 10:    proc  0 (:init::1) Question1_template.pml:41 (state 6)
5 processes created
Exit-Status 0
```

```
[queues, step  10]

q 1  :: (up[0]):
q 2  :: (station(1):in)
:
q 3  :: (down[0]):
q 4  :: (down[1]):
```

The above error trail can be found in Question1_template.pml.trail (fully reproduced in Appendix). We can observe that P3 starts sending right after P4 starts sending. Both stations are sending at the same time, resulting in the deadlock described above.

There is also an additional problem with the Promela model. By allowing an infinite amount of data to be sent in one round of communication, the property can be trivially violated by simply never ending the communication.

Thus, there are two key augmentations to the protocol:

1. The first augmentation is the addition of a maximum number of data packets that can be sent. This change is localised to the send operation. The maximum number 3 is chosen arbitrarily.

Before	**After**

```
// sending
do
  ::out!data->in?data
  ::out!stop->break
od;
```
	```
	// sending
	int count = 3;
	do
	  ::count > 0->
	    out!data;
	    in?data;
	```

	`    count--;`
	`  ::out!stop->break`
	`od;`

2. The second, more meaningful augmentation is introduced to synchronise the two stations without the use of global state variables, which cannot be modelled in the real world. This is done by introducing the concept of "turns", where each station can only send data when it is their turn. Each station maintains their own concept of turns, implicitly synchronised once during initialisation time. The turns are switched by a successful round of sending/receiving, which is the only way two stations can communicate in real life. This eliminates the possibility that an opposite station will try to send just as a station is sending, resolving the deadlock.

Before	**After**

	`// additional declaration`
	`int turn[2];`

`// receiving`
`out!stop;`
`busy[id]=false`
	`// receiving`
	`out!stop;`
	`turn[id] = 1 - turn[id];`
	`busy[id]=false;`

`// sending`
`::atomic{!busy[id]->busy[id]=true};`
`...`
`in?stop;`
`busy[id]=false`
	`// sending`
	`::atomic{!busy[id] && turn[id] == id`
	`-> busy[id]=true};`
	`...`
	`in?stop;`
	`turn[id] = 1 - turn[id];`
	`busy[id]=false;`

The augmented code can be found in Question1.pml.

2. Reassembly Deadlock

2.1 Promela Model

The Promela code modelling this system can be found in Question2.pml. Note that the size of the buffer (represented with an internal channel) is set to be an arbitrary value of 6. 3 is the minimum size the buffer can take on, as that is the minimum number of messages that is needed to reassemble an output message.

Also note that instead of using a Promela struct to represent the reassembled output type, I use an mtype. This does not make a meaningful difference in modelling the system or in checking the LTL property.

2.2 Deadlock Explanation

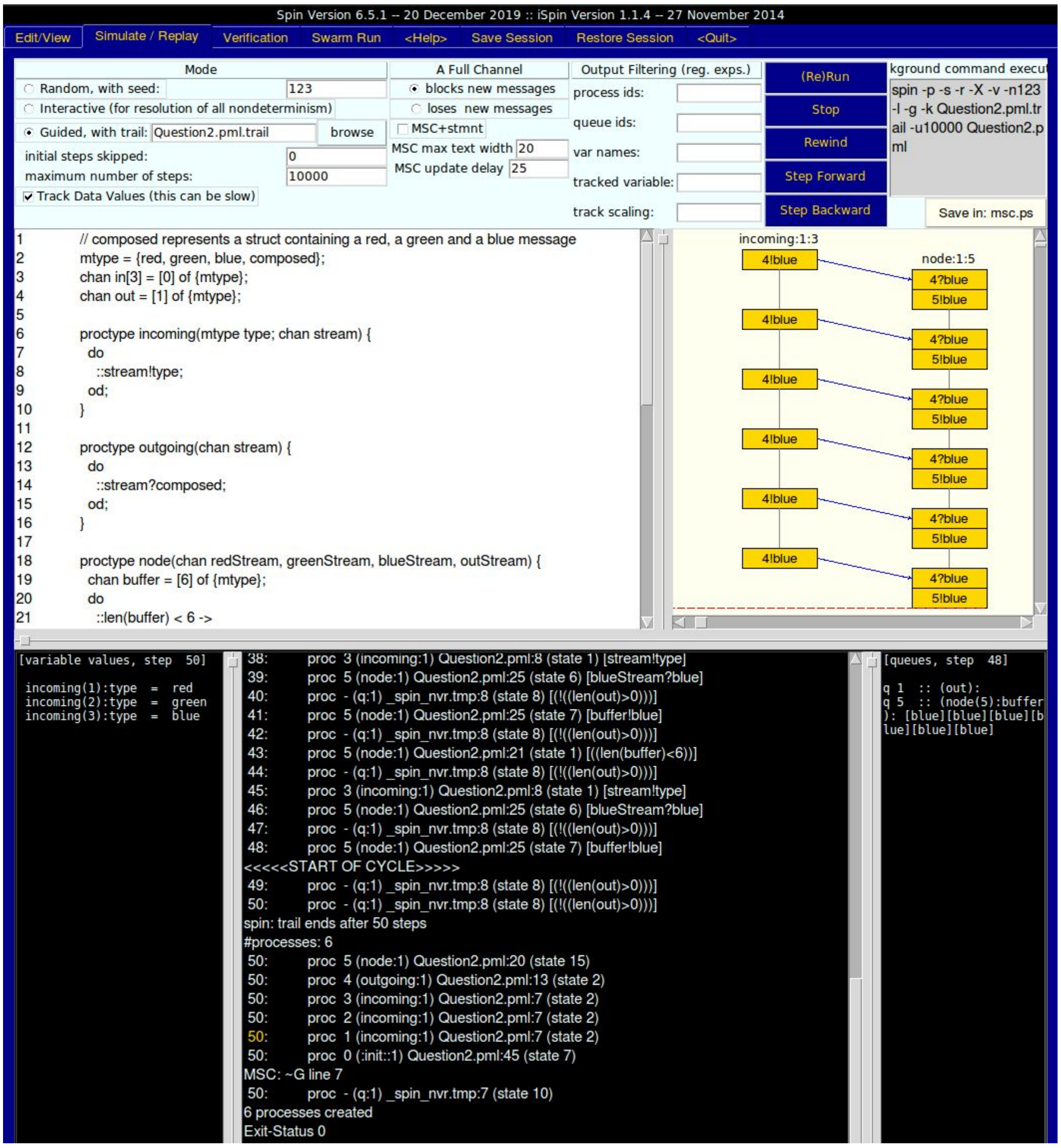
The reassembly deadlock can occur when the buffer becomes full with less than all 3 types of messages. This means that the output message cannot be reassembled, and the missing type(s) cannot be accepted into the buffer as it is now full. The node cannot perform any further action, resulting in a deadlock.

2.3. LTL and Counterexample

The LTL property used is as follows:

```
ltl q {[](<>(len(out) > 0))}
```

This property states that the length of the output channel, which receives the reassembled messages, will always eventually have a length greater than 0. This is akin to saying that the channel should continually receive reassembled messages. For that to happen, no deadlock can occur during the process of reassembly.



Mode
Random, with seed: 123
Interactive (for resolution of all nondeterminism)
[✓ Guided, with trail: Question2.pml.trail]
initial steps skipped: 0
maximum number of steps: 10000
[✓ Track Data Values (this can be slow)]

A Full Channel
[✓ blocks new messages]
loses new messages
MSC+stmnt
MSC max text width 20
MSC update delay 25

Output Filtering (reg. exps.)
process ids:
queue ids:
var names:
tracked variable:
track scaling:

kground command execu
spin -p -s -r -X -v -n123 -l -g -k Question2.pml.trail -u10000 Question2.pml

Save in: msc.ps

```
// composed represents a struct containing a red, a green and a blue message
mtype = {red, green, blue, composed};
chan in[3] = [0] of {mtype};
chan out = [1] of {mtype};

proctype incoming(mtype type; chan stream) {
  do
    ::stream!type;
  od;
}

proctype outgoing(chan stream) {
  do
    ::stream?composed;
  od;
}

proctype node(chan redStream, greenStream, blueStream, outStream) {
  chan buffer = [6] of {mtype};
  do
    ::len(buffer) < 6 ->
```

```
[variable values, step  50]

 incoming(1):type  =  red
 incoming(2):type  =  green
 incoming(3):type  =  blue
```

```
38:      proc  3 (incoming:1) Question2.pml:8 (state 1) [stream!type]
39:      proc  5 (node:1) Question2.pml:25 (state 6) [blueStream?blue]
40:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
41:      proc  5 (node:1) Question2.pml:25 (state 7) [buffer!blue]
42:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
43:      proc  5 (node:1) Question2.pml:21 (state 1) [((len(buffer)<6))]
44:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
45:      proc  3 (incoming:1) Question2.pml:8 (state 1) [stream!type]
46:      proc  5 (node:1) Question2.pml:25 (state 6) [blueStream?blue]
47:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
48:      proc  5 (node:1) Question2.pml:25 (state 7) [buffer!blue]
<<<<<START OF CYCLE>>>>>
49:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
50:      proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((len(out)>0)))]
spin: trail ends after 50 steps
#processes: 6
50:      proc  5 (node:1) Question2.pml:20 (state 15)
50:      proc  4 (outgoing:1) Question2.pml:13 (state 2)
50:      proc  3 (incoming:1) Question2.pml:7 (state 2)
50:      proc  2 (incoming:1) Question2.pml:7 (state 2)
50:      proc  1 (incoming:1) Question2.pml:7 (state 2)
50:      proc  0 (:init::1) Question2.pml:45 (state 7)
MSC: ~G line 7
50:      proc  - (q:1) _spin_nvr.tmp:7 (state 10)
6 processes created
Exit-Status 0
```

```
[queues, step  48]

q 1  :: (out):
q 5  :: (node(5):buffer
): [blue][blue][blue][b
lue][blue][blue]
```

The counterexample trace can be found in Question2.pml.trail (fully reproduced in Appendix). In essence, the node receives messages from the blue incoming stream until the buffer is full. Since the buffer is full, it is unable to receive messages of the other types. Reassembly cannot occur and no messages can be removed from the buffer, resulting in the deadlock.

3. Weather System

3.1. Promela Model

The Promela code modelling this system can be found in Question3.pml. Global state variables are used to model state, while channels are used to model communications (such as informing a client to get weather).

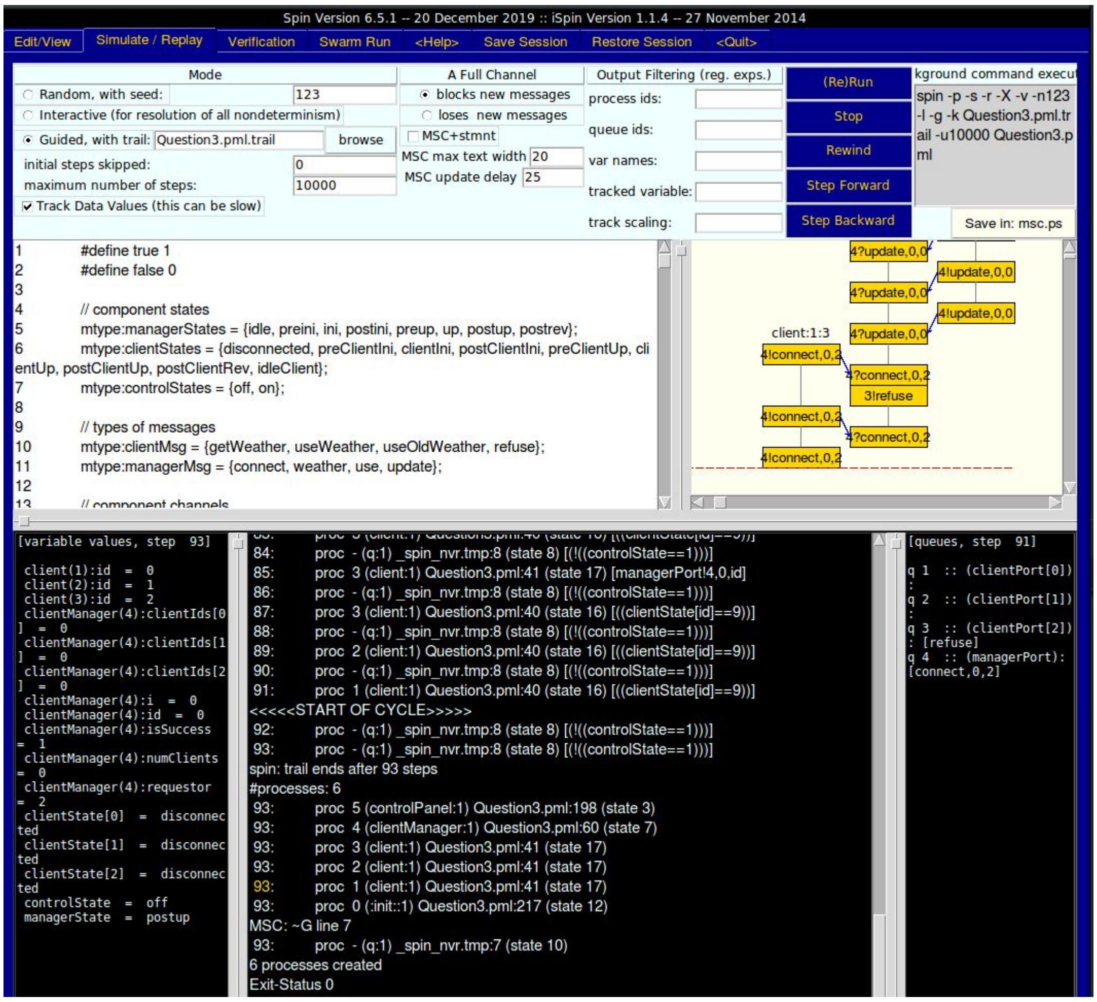
3.2 LTL Property

The LTL property used is as follows:

```
ltl q {[](controlState == off -> <>(controlState == on))}
```

As controlState holds the WCP state, the property states that whenever a WCP is disabled, it will be enabled again. For this property to be true, it means that all updates that are started (which will cause WCP to be disabled) will result in completion (which will re-enable the WCP).

3.3. Deadlock



```
Mode
Random, with seed: 123
Interactive (for resolution of all nondeterminism)
Guided, with trail: Question3.pml.trail
initial steps skipped: 0
maximum number of steps: 10000
[✓] Track Data Values (this can be slow)

A Full Channel
(●) blocks new messages
( ) loses new messages
[ ] MSC+stmnt
MSC max text width 20
MSC update delay 25

Output Filtering (reg. exps.)
process ids:
queue ids:
var names:
tracked variable:
track scaling:

kground command execu
spin -p -s -r -X -v -n123
-l -g -k Question3.pml.tr
ail -u10000 Question3.p
ml
```

```
#define true 1
#define false 0

// component states
mtype:managerStates = {idle, preini, ini, postini, preup, up, postup, postrev};
mtype:clientStates = {disconnected, preClientIni, clientIni, postClientIni, preClientUp, cli
entUp, postClientUp, postClientRev, idleClient};
mtype:controlStates = {off, on};

// types of messages
mtype:clientMsg = {getWeather, useWeather, useOldWeather, refuse};
mtype:managerMsg = {connect, weather, use, update};

// component channels
```

```
[variable values, step  93]

 client(1):id  =  0
 client(2):id  =  1
 client(3):id  =  2
 clientManager(4):clientIds[0
]  =  0
 clientManager(4):clientIds[1
]  =  0
 clientManager(4):clientIds[2
]  =  0
 clientManager(4):i  =  0
 clientManager(4):id  =  0
 clientManager(4):isSuccess
=  1
 clientManager(4):numClients
=  0
 clientManager(4):requestor
=  2
 clientState[0]  =  disconnec
ted
 clientState[1]  =  disconnec
ted
 clientState[2]  =  disconnec
ted
 controlState  =  off
 managerState  =  postup
```

```
 84:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
 85:     proc  3 (client:1) Question3.pml:41 (state 17) [managerPort!4,0,id]
 86:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
 87:     proc  3 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 88:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
 89:     proc  2 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 90:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
 91:     proc  1 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
<<<<<START OF CYCLE>>>>>
 92:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
 93:     proc  - (q:1) _spin_nvr.tmp:8 (state 8) [(!((controlState==1)))]
spin: trail ends after 93 steps
#processes: 6
 93:     proc  5 (controlPanel:1) Question3.pml:198 (state 3)
 93:     proc  4 (clientManager:1) Question3.pml:60 (state 7)
 93:     proc  3 (client:1) Question3.pml:41 (state 17)
 93:     proc  2 (client:1) Question3.pml:41 (state 17)
 93:     proc  1 (client:1) Question3.pml:41 (state 17)
 93:     proc  0 (:init::1) Question3.pml:217 (state 12)
MSC: ~G line 7
 93:     proc  - (q:1) _spin_nvr.tmp:7 (state 10)
6 processes created
Exit-Status 0
```

```
[queues, step  91]

q 1  :: (clientPort[0])
:
q 2  :: (clientPort[1])
:
q 3  :: (clientPort[2])
: [refuse]
q 4  :: (managerPort):
[connect,0,2]
```

There exists a counterexample (reproduced fully in Appendix and found in Question3.pml.trail) where the LTL property does not hold true. If the client manager is transitioning between states during initialization/update, and a client repeatedly requests to connect, the client manager may be stuck in an infinite loop refusing these requests. In the above example, the client manager is stuck in post updating state and is unable to complete the update operation.

There also exists a deadlock when a client fails to get new weather during initialization. The client manager goes idle without enabling the WCP, preventing updates from being performed entirely.

3.4. Model Augmentation

The augmented model can be found in Question3_modified.pml. The augmentations are summarised as follows:

1. Having the CM re-enable the WCP if the connecting client fails to get new weather. This will resolve the second bug described in Section 3.3.

2. The refuse message is removed. Clients who wish to connect will wait indefinitely for the connect request to be accepted. Size of the manager port channel is increased accordingly to let these connect requests sit with remaining space in the channel, and receive operations are modified to read from anywhere in the channel, not just the front (?? syntax instead of ?).

3. In addition, the client manager only entertains connection and update requests when it is idle

4. A separate channel is used for WCP -> CM communications.

Appendix

Counterexample Trace for 1.2

```
spin: Question1_template.pml:48, Error: syntax error   saw 'operator: <'
using statement merging
Starting station with pid 1
  1:    proc  0 (:init::1) Question1_template.pml:36 (state 1)   [(run station(0,up[1],down[1]))]
Starting station with pid 2
  2:    proc  0 (:init::1) Question1_template.pml:37 (state 2)   [(run station(1,up[0],down[0]))]
Starting station with pid 3
  3:    proc  0 (:init::1) Question1_template.pml:38 (state 3)   [(run station(0,down[0],up[0]))]
Starting station with pid 4
  4:    proc  0 (:init::1) Question1_template.pml:39 (state 4)   [(run station(1,down[1],up[1]))]
  5:    proc  4 (station:1) Question1_template.pml:22 (state 15)        [(!(busy[id]))]
  5:    proc  4 (station:1) Question1_template.pml:22 (state 16)        [busy[id] = 1]
  6:    proc  4 (station:1) Question1_template.pml:23 (state 18)        [out!start]
  7:    proc  3 (station:1) Question1_template.pml:22 (state 15)        [(!(busy[id]))]
  7:    proc  3 (station:1) Question1_template.pml:22 (state 16)        [busy[id] = 1]
  8:    proc  3 (station:1) Question1_template.pml:23 (state 18)        [out!start]
  9:    proc  2 (station:1) Question1_template.pml:13 (state 1)         [in?start]
 10:    proc  1 (station:1) Question1_template.pml:13 (state 1)         [in?start]
spin: trail ends after 10 steps
#processes: 5
 10:    proc  4 (station:1) Question1_template.pml:24 (state 19)
 10:    proc  3 (station:1) Question1_template.pml:24 (state 19)
 10:    proc  2 (station:1) Question1_template.pml:14 (state 4)
 10:    proc  1 (station:1) Question1_template.pml:14 (state 4)
 10:    proc  0 (:init::1) Question1_template.pml:41 (state 6)
5 processes created
Exit-Status 0
```

Counterexample Trace for 2.3

```
ltl q: [] (<> ((len(out)>0)))
starting claim 4
using statement merging
MSC: ~G line 3
  1:    proc  - (q:1) _spin_nvr.tmp:3 (state 1)         [(!((len(out)>0)))]
Never claim moves to line 3         [(!((len(out)>0)))]
Starting incoming with pid 2
  2:    proc  0 (:init::1) Question2.pml:39 (state 1)    [(run incoming(red,in[0]))]
Starting incoming with pid 3
  3:    proc  0 (:init::1) Question2.pml:40 (state 2)    [(run incoming(green,in[1]))]
Starting incoming with pid 4
  4:    proc  0 (:init::1) Question2.pml:41 (state 3)    [(run incoming(blue,in[2]))]
Starting outgoing with pid 5
  5:    proc  0 (:init::1) Question2.pml:42 (state 4)    [(run outgoing(out))]
Starting node with pid 6
  6:    proc  0 (:init::1) Question2.pml:43 (state 5)    [(run node(in[0],in[1],in[2],out))]
MSC: ~G line 8
  7:    proc  - (q:1) _spin_nvr.tmp:8 (state 8)         [(!((len(out)>0)))]
```

```
Never claim moves to line 8            [(!((len(out)>0)))]
  8:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
  9:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 10:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 11:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 12:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 13:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
 14:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 15:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
 16:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 17:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 18:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 19:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 20:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
 21:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 22:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
 23:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 24:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 25:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 26:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 27:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
 28:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 29:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
 30:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 31:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 32:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 33:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 34:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
 35:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 36:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
 37:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 38:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 39:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 40:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 41:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
 42:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 43:        proc  5 (node:1) Question2.pml:21 (state 1)   [((len(buffer)<6))]
 44:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 45:        proc  3 (incoming:1) Question2.pml:8 (state 1)           [stream!type]
 46:        proc  5 (node:1) Question2.pml:25 (state 6)   [blueStream?blue]
 47:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 48:        proc  5 (node:1) Question2.pml:25 (state 7)   [buffer!blue]
<<<<<START OF CYCLE>>>>>
 49:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
 50:        proc  - (q:1) _spin_nvr.tmp:8 (state 8)       [(!((len(out)>0)))]
spin: trail ends after 50 steps
#processes: 6
 50:        proc  5 (node:1) Question2.pml:20 (state 15)
 50:        proc  4 (outgoing:1) Question2.pml:13 (state 2)
 50:        proc  3 (incoming:1) Question2.pml:7 (state 2)
 50:        proc  2 (incoming:1) Question2.pml:7 (state 2)
 50:        proc  1 (incoming:1) Question2.pml:7 (state 2)
 50:        proc  0 (:init::1) Question2.pml:45 (state 7)
MSC: ~G line 7
 50:        proc  - (q:1) _spin_nvr.tmp:7 (state 10)
6 processes created
Exit-Status 0
```

Counterexample Trace for 3.3

```
ltl q: [] ((! ((controlState==2))) || (<> ((controlState==1))))
```

```
starting claim 4
using statement merging
MSC: ~G line 4
  1:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
Never claim moves to line 4	[(1)]
  2:	proc  0 (:init::1) Question3.pml:206 (state 1)	[clientState[0] = 9]
  2:	proc  0 (:init::1) Question3.pml:207 (state 2)	[clientState[1] = 9]
  2:	proc  0 (:init::1) Question3.pml:208 (state 3)	[clientState[2] = 9]
  2:	proc  0 (:init::1) Question3.pml:209 (state 4)	[managerState = 8]
  2:	proc  0 (:init::1) Question3.pml:210 (state 5)	[controlState = 1]
Starting client with pid 2
  3:	proc  0 (:init::1) Question3.pml:211 (state 6)	[(run client(0,clientPort[0]))]
Starting client with pid 3
  4:	proc  0 (:init::1) Question3.pml:212 (state 7)	[(run client(1,clientPort[1]))]
Starting client with pid 4
  5:	proc  0 (:init::1) Question3.pml:213 (state 8)	[(run client(2,clientPort[2]))]
Starting clientManager with pid 5
  6:	proc  0 (:init::1) Question3.pml:214 (state 9)	[(run clientManager())]
Starting controlPanel with pid 6
  7:	proc  0 (:init::1) Question3.pml:215 (state 10)	[(run controlPanel())]
  8:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
  9:	proc  5 (controlPanel:1) Question3.pml:199 (state 1)	[((controlState==1))]
 10:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 11:	proc  5 (controlPanel:1) Question3.pml:200 (state 2)	[managerPort!1,0,0]
 12:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 13:	proc  5 (controlPanel:1) Question3.pml:199 (state 1)	[((controlState==1))]
 14:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 15:	proc  4 (clientManager:1) Question3.pml:62 (state 10)	[managerPort?1,_,_]
 16:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 17:	proc  5 (controlPanel:1) Question3.pml:200 (state 2)	[managerPort!1,0,0]
 18:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 19:	proc  5 (controlPanel:1) Question3.pml:199 (state 1)	[((controlState==1))]
 20:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 21:	proc  4 (clientManager:1) Question3.pml:64 (state 11)	[((managerState==8))]
 22:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 23:	proc  4 (clientManager:1) Question3.pml:65 (state 12)	[i = 0]
 24:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 25:	proc  4 (clientManager:1) Question3.pml:68 (state 17)	[else]
 26:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 27:	proc  4 (clientManager:1) Question3.pml:69 (state 22)	[managerState = 4]
 28:	proc  - (q:1) _spin_nvr.tmp:4 (state 3)	[(1)]
 29:	proc  4 (clientManager:1) Question3.pml:70 (state 23)	[controlState = 2]
MSC: ~G line 3
 30:	proc  - (q:1) _spin_nvr.tmp:3 (state 1)	[((!(!((controlState==2)))&&!((controlState==1))))]
Never claim moves to line 3	[((!(!((controlState==2)))&&!((controlState==1))))]
 31:	proc  4 (clientManager:1) Question3.pml:62 (state 10)	[managerPort?1,_,_]
MSC: ~G line 8
 32:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
Never claim moves to line 8	[(!((controlState==1)))]
 33:	proc  5 (controlPanel:1) Question3.pml:200 (state 2)	[managerPort!1,0,0]
 34:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 35:	proc  4 (clientManager:1) Question3.pml:71 (state 24)	[else]
 36:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 37:	proc  4 (clientManager:1) Question3.pml:62 (state 10)	[managerPort?1,_,_]
 38:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 39:	proc  4 (clientManager:1) Question3.pml:71 (state 24)	[else]
 40:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 41:	proc  4 (clientManager:1) Question3.pml:108 (state 57)	[((managerState==4))]
 42:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 43:	proc  4 (clientManager:1) Question3.pml:109 (state 58)	[i = 0]
 44:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 45:	proc  4 (clientManager:1) Question3.pml:112 (state 63)	[else]
 46:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 47:	proc  4 (clientManager:1) Question3.pml:113 (state 68)	[i = 0]
 48:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 49:	proc  4 (clientManager:1) Question3.pml:116 (state 73)	[else]
 50:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 51:	proc  4 (clientManager:1) Question3.pml:117 (state 78)	[managerState = 3]
 52:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 53:	proc  4 (clientManager:1) Question3.pml:118 (state 79)	[((managerState==3))]
 54:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 55:	proc  4 (clientManager:1) Question3.pml:120 (state 80)	[isSuccess = 1]
 55:	proc  4 (clientManager:1) Question3.pml:120 (state 81)	[i = 0]
 56:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 57:	proc  4 (clientManager:1) Question3.pml:127 (state 90)	[else]
 58:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 59:	proc  4 (clientManager:1) Question3.pml:129 (state 95)	[((isSuccess==1))]
 59:	proc  4 (clientManager:1) Question3.pml:130 (state 96)	[i = 0]
 60:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 61:	proc  4 (clientManager:1) Question3.pml:134 (state 102)	[else]
 62:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 63:	proc  4 (clientManager:1) Question3.pml:135 (state 107)	[managerState = 2]
 64:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 65:	proc  3 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 66:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 67:	proc  3 (client:1) Question3.pml:41 (state 17) [managerPort!4,0,id]
 68:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 69:	proc  4 (clientManager:1) Question3.pml:53 (state 1)	[managerPort?4,_,requestor]
 70:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 71:	proc  4 (clientManager:1) Question3.pml:59 (state 6)	[else]
 72:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 73:	proc  4 (clientManager:1) Question3.pml:60 (state 7)	[clientPort[requestor]!1]
 74:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 75:	proc  3 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 76:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 77:	proc  3 (client:1) Question3.pml:41 (state 17) [managerPort!4,0,id]
 78:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 79:	proc  4 (clientManager:1) Question3.pml:53 (state 1)	[managerPort?4,_,requestor]
 80:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 81:	proc  4 (clientManager:1) Question3.pml:59 (state 6)	[else]
 82:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 83:	proc  3 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 84:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 85:	proc  3 (client:1) Question3.pml:41 (state 17) [managerPort!4,0,id]
 86:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 87:	proc  3 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 88:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 89:	proc  2 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
 90:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 91:	proc  1 (client:1) Question3.pml:40 (state 16) [((clientState[id]==9))]
<<<<<START OF CYCLE>>>>>
 92:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
 93:	proc  - (q:1) _spin_nvr.tmp:8 (state 8)	[(!((controlState==1)))]
spin: trail ends after 93 steps
#processes: 6
 93:	proc  5 (controlPanel:1) Question3.pml:198 (state 3)
 93:	proc  4 (clientManager:1) Question3.pml:60 (state 7)
 93:	proc  3 (client:1) Question3.pml:41 (state 17)
 93:	proc  2 (client:1) Question3.pml:41 (state 17)
 93:	proc  1 (client:1) Question3.pml:41 (state 17)
 93:	proc  0 (:init::1) Question3.pml:217 (state 12)
MSC: ~G line 7
```

```
 93:        proc  - (q:1) _spin_nvr.tmp:7 (state 10)
6 processes created
Exit-Status 0
```