Question 1

1. Code can be found in problem-1.smv
2. We use the following LTL property:

LTLSPEC G ((up0 = start -> F (up0 = stop)) & (up1 = start -> F (up1 = stop)) & (down0 = start -> F (down0 = stop)) & (down1 = start -> F (down1 = stop)))

This property holds if any communication that is started (with a start signal) is finished (with a stop signal)

When we run problem-1.smv with NuSMV, we obtain that the specified property is false as demonstrated by the following execution sequence

Trace Description: LTL Counterexample

Trace Type: Counterexample

-> State: 1.1 <-

busy[0] = FALSE

busy[1] = FALSE

counter = 0

up0 = start

down0 = start

up1 = start

down1 = start

-> Input: 1.2 <-

\_process\_selector\_ = operator\_1

running = FALSE

operator\_4.running = FALSE

operator\_3.running = FALSE

operator\_2.running = FALSE

operator\_1.running = TRUE

-> State: 1.2 <-

busy[0] = TRUE

-> Input: 1.3 <-

\_process\_selector\_ = operator\_2

operator\_2.running = TRUE

operator\_1.running = FALSE

-> State: 1.3 <-

busy[1] = TRUE

-> Input: 1.4 <-

\_process\_selector\_ = operator\_3

operator\_3.running = TRUE

operator\_2.running = FALSE

-> State: 1.4 <-

up0 = ack

-> Input: 1.5 <-

\_process\_selector\_ = operator\_4

operator\_4.running = TRUE

operator\_3.running = FALSE

-> State: 1.5 <-

up1 = ack

-> Input: 1.6 <-

\_process\_selector\_ = operator\_2

operator\_4.running = FALSE

operator\_2.running = TRUE

-> State: 1.6 <-

down0 = data

-> Input: 1.7 <-

-> State: 1.7 <-

counter = 1

-> Input: 1.8 <-

\_process\_selector\_ = operator\_1

operator\_2.running = FALSE

operator\_1.running = TRUE

-> State: 1.8 <-

down1 = data

-> Input: 1.9 <-

\_process\_selector\_ = operator\_2

operator\_2.running = TRUE

operator\_1.running = FALSE

-> State: 1.9 <-

counter = 2

-> Input: 1.10 <-

\_process\_selector\_ = operator\_3

operator\_3.running = TRUE

operator\_2.running = FALSE

-> State: 1.10 <-

up0 = data

-> Input: 1.11 <-

\_process\_selector\_ = operator\_4

operator\_4.running = TRUE

operator\_3.running = FALSE

-> State: 1.11 <-

up1 = data

-> Input: 1.12 <-

\_process\_selector\_ = main

running = TRUE

operator\_4.running = FALSE

-> State: 1.12 <-

-> Input: 1.13 <-

\_process\_selector\_ = operator\_1

running = FALSE

operator\_1.running = TRUE

-- Loop starts here

-> State: 1.13 <-

counter = 3

-> Input: 1.14 <-

\_process\_selector\_ = operator\_2

operator\_2.running = TRUE

operator\_1.running = FALSE

-- Loop starts here

-> State: 1.14 <-

-> Input: 1.15 <-

\_process\_selector\_ = operator\_3

operator\_3.running = TRUE

operator\_2.running = FALSE

-- Loop starts here

-> State: 1.15 <-

-> Input: 1.16 <-

\_process\_selector\_ = operator\_4

operator\_4.running = TRUE

operator\_3.running = FALSE

-- Loop starts here

-> State: 1.16 <-

-> Input: 1.17 <-

\_process\_selector\_ = main

running = TRUE

operator\_4.running = FALSE

-- Loop starts here

-> State: 1.17 <-

-> Input: 1.18 <-

\_process\_selector\_ = operator\_1

running = FALSE

operator\_1.running = TRUE

-- Loop starts here

-> State: 1.18 <-

-> Input: 1.19 <-

\_process\_selector\_ = operator\_2

operator\_2.running = TRUE

operator\_1.running = FALSE

-- Loop starts here

-> State: 1.19 <-

-> Input: 1.20 <-

\_process\_selector\_ = operator\_3

operator\_3.running = TRUE

operator\_2.running = FALSE

-- Loop starts here

-> State: 1.20 <-

-> Input: 1.21 <-

\_process\_selector\_ = operator\_4

operator\_4.running = TRUE

operator\_3.running = FALSE

-- Loop starts here

-> State: 1.21 <-

-> Input: 1.22 <-

\_process\_selector\_ = main

running = TRUE

operator\_4.running = FALSE

-> State: 1.22 <-

Question 2

1. Code can be found in problem-2.smv

In order to verify that the solution preserves mutual exclusion, we use the following LTL property:

LTLSPEC G!((proc\_1.key = TRUE) & (proc\_2.key = TRUE))

When we run problem-2.smv with NuSMV, we obtain that the specified property is true.

1. This solution is not a good solution to the mutual exclusion problem as livelock/deadlock can occur. In order to verify this, we use the following LTL property:

LTLSPEC G(F(proc\_1.key = TRUE) & F(proc\_2.key = TRUE))

When we run problem-2.smv with NuSMV, we obtain that the specified property is false as demonstrated by the following execution sequence

Trace Description: LTL Counterexample

Trace Type: Counterexample

-> State: 1.1 <-

lock = TRUE

proc\_1.key = FALSE

proc\_2.key = FALSE

-> Input: 1.2 <-

\_process\_selector\_ = proc\_1

running = FALSE

proc\_2.running = FALSE

proc\_1.running = TRUE

-- Loop starts here

-> State: 1.2 <-

lock = FALSE

-> Input: 1.3 <-

\_process\_selector\_ = proc\_2

proc\_2.running = TRUE

proc\_1.running = FALSE

-- Loop starts here

-> State: 1.3 <-

-> Input: 1.4 <-

\_process\_selector\_ = main

running = TRUE

proc\_2.running = FALSE

-- Loop starts here

-> State: 1.4 <-

-> Input: 1.5 <-

\_process\_selector\_ = proc\_1

running = FALSE

proc\_1.running = TRUE

-- Loop starts here

-> State: 1.5 <-

-> Input: 1.6 <-

\_process\_selector\_ = proc\_2

proc\_2.running = TRUE

proc\_1.running = FALSE

-- Loop starts here

-> State: 1.6 <-

-> Input: 1.7 <-

\_process\_selector\_ = main

running = TRUE

proc\_2.running = FALSE

-> State: 1.7 <-

This solution is also not a good solution to the mutual exclusion problem as starvation can occur. In order to verify this, we use the following LTL property:

LTLSPEC G(F(proc\_1.key = TRUE) | F(proc\_2.key = TRUE))

When we run problem-2.smv with NuSMV, we obtain that the specified property is false as demonstrated by the following execution sequence

Trace Description: LTL Counterexample

Trace Type: Counterexample

-> State: 2.1 <-

lock = TRUE

proc\_1.key = FALSE

proc\_2.key = FALSE

-> Input: 2.2 <-

\_process\_selector\_ = proc\_1

running = FALSE

proc\_2.running = FALSE

proc\_1.running = TRUE

-- Loop starts here

-> State: 2.2 <-

lock = FALSE

-> Input: 2.3 <-

\_process\_selector\_ = proc\_2

proc\_2.running = TRUE

proc\_1.running = FALSE

-- Loop starts here

-> State: 2.3 <-

-> Input: 2.4 <-

\_process\_selector\_ = main

running = TRUE

proc\_2.running = FALSE

-- Loop starts here

-> State: 2.4 <-

-> Input: 2.5 <-

\_process\_selector\_ = proc\_1

running = FALSE

proc\_1.running = TRUE

-- Loop starts here

-> State: 2.5 <-

-> Input: 2.6 <-

\_process\_selector\_ = proc\_2

proc\_2.running = TRUE

proc\_1.running = FALSE

-- Loop starts here

-> State: 2.6 <-

-> Input: 2.7 <-

\_process\_selector\_ = main

running = TRUE

proc\_2.running = FALSE

-> State: 2.7 <-

Question 3

1. Code can be found in problem-3.smv

In order to verify that the solution preserves mutual exclusion, we use the following LTL property:

LTLSPEC G !((flag[0] = TRUE & (turn = 0 | flag[1] = FALSE)) & (flag[1] = TRUE & (turn = 1 | flag[0] = FALSE)))

When we run problem-3.smv with NuSMV, we obtain that the specified property is true.

1. In order to verify that the solution is starvation-free, we use the following LTL property:

LTLSPEC G(((turn = 1 | flag[0] = FALSE) & flag[1] = TRUE) -> F(turn = 0 | flag[1] = FALSE) & (((turn = 0 | flag[1] = FALSE) & flag[0] = TRUE) -> F(turn = 1 | flag[0] = FALSE)))

When we run problem-3.smv with NuSMV, we obtain that the specified property is true.

This is because all processes that enter the loop eventually exit it (assuming they don't stay in the critical section indefinitely).