F21DS1
Sample Exam Questions.
(A. Ireland)

Q.1

(a) The process of formal verification can be viewed in terms of a number of phases. Identify and describe in detail the three core phases associated with formal verification.

(9)

(b)

The Automatic Warning System (**AWS**) is designed to reduce the risk of a train driver passing danger signals. The **AWS** is a reactive system that maintains an ongoing interaction with the:

- track-side signals;
- train driver;
- braking system of the train.

The **AWS** operates as follows:

- When a train passes a **track-side signal** a message is sent from the signal to the **AWS** indicating whether a **danger** or **proceed** aspect is being displayed.

- If the **AWS** receives a **proceed** aspect from the **track-side signal** then a **bell** message is sent to the **train driver** while a **horn** message is sent if a **danger** aspect is received.

- On receiving a **bell** message from the **AWS** the **train driver** does nothing, but keeps monitoring messages from the **AWS**. However, on receiving a **horn** message from the **AWS** the **train driver** should send an **ack** (acknowledgement) message back to the **AWS**.

Cont/Q1

...Cont/

- On receiving an **ack** message from the **train driver** the **AWS** continues to monitor messages from the **track-side signals**. However, if no **ack** message is received from the **train driver** then eventually a time-out will take place within the **AWS**, at which point it will send an **apply-brakes** message to the train's **braking system**.

Your task is to model in Promela the **AWS** system and its environment as described above. You should construct your model so that each of the distributed components that make up the overall system are represented as distinct non-terminating processes. You should model all **track-side signals** as a single process that generates a non-terminating sequence of signal aspects. In addition, you should model the **train driver** so that on receiving a **horn** message from the **AWS** it will non-deterministically decide whether or not to send an **ack** (acknowledgement) message.

(16)

Q2.

(a) Road traffic signals are required in order to prevent collisions at the point where two or more roads cross. Consider a road junction involving two roads which cross, one carrying traffic to the north and the other carrying traffic to the west. In order to prevent collisions two traffic signals are required, i.e. one to control traffic travelling from south to north and another to control traffic travelling from east to west. The following is an attempt to model the traffic signals at such a road junction in Promela:

```
mtype = {green, red};
mtype north_aspect, west_aspect;
proctype north_signal()
{   do
    :: north_aspect = red;
    :: north_aspect = green;
    od
}
proctype west_signal()
{     do
      :: west_aspect = green;
      :: west_aspect = red;
      od
}
init {     atomic{ run north_signal(); run west_signal()}}
```

...Cont/

...Cont/Q Q, 2

Note that the global variables north_aspect and west_aspect record the colour of signal that is displayed to the north bound and west bound traffic respectively. Each of these variables can take one of two values, i.e. red indicating that cars should stop, or green indicating that cars should proceed. Now given the following property:

"at no point in time will cars travelling north and cars travelling west both be given a proceed signal."

attempt the following two specification tasks:

- Firstly, extend the model by introducing a monitor process that expresses the property given above. Using the SPIN simulator what effect would your monitor process have on the execution of the traffic signal model?

(5)

- Secondly, using the SPIN verifier, the model could be verified with respect to the given property in 2 modes, i.e. it could be verified as either a **desired behaviour** or as an **error behaviour**. Construct linear time temporal logic formulae that express the given property in terms of each of these verification modes.

(5)

(b) Extend the traffic signal model with a process called controller that will ensure that the property specified above holds. You should define controller such that:

- all communication with the processes north_signal and west_signal is achieved by means of 2 distinct message channels.
- all communications should be **synchronous**.

Note that the extension to the model will require modifications to both the north_signal and west_signal process definitions.

(15)

END OF PAPER

Q.1

a)

Modelling Phase – construction of an abstraction of the real system. Care must be taken to ensure no critical details of the system are lost via abstraction.

Specification Phase – description of desirable or error behaviours, e.g. temporal properties.

Analysis Phase – ensuring that model satisfies the specification, e.g.

Simulation ($\hat{=}$ testing)

Model checking (finite state systems)

Theorem proving (infinite state systems)

QUES NO.: Q1 ~~[illegible]~~	EXAM DIET:
1(b)	

```
mtype = {proceed, danger, bell, horn, ack, brake};

chan signal_to_aws = [0] of { mtype }; ✓
chan aws_to_driver = [0] of { mtype }; ✓
chan driver_to_aws = [0] of { mtype }; ✓
chan aws_to_brakes = [0] of { mtype }; ✓

proctype aws(chan from_sig, to_driver, from_driver, brakes)
{
       do
       :: from_sig?proceed; to_driver!bell;
       :: from_sig?danger;  to_driver!horn;
                             if
                             :: from_driver?ack;
                             :: timeout -> brakes!brake;
                             fi
       od
}

proctype driver(chan from_aws, to_aws)
{
       do
       :: from_aws?horn; if ::to_aws!ack; ::skip; fi
       :: from_aws?bell;
       od
}

proctype signals(chan to_aws)
{
       do
       :: to_aws!danger;
       :: to_aws!proceed;
       od
}

proctype brakes(chan from_aws)
{
       do
       :: from_aws?brake;
       od
}

init {  atomic{

       run signals(signal_to_aws);
       run aws(signal_to_aws, aws_to_driver, driver_to_aws, aws_to_brakes);
       run driver(aws_to_driver, driver_to_aws);
       run brakes(aws_to_brakes)
       }
}
```

QUES NO.: Q1 & 2	EXAM DIET:
2(a)	

```
mtype = {green, red};

mtype north_aspect, west_aspect;

proctype north_signal()
{
        do
        :: north_aspect = red;
        :: north_aspect = green;
        od
}

proctype west_signal()
{
        do
        :: west_aspect = green;
        :: west_aspect = red;
        od
}

proctype monitor()
{
        do
        ::assert(!(north_aspect == green &&west_aspect  == green))
        od
}

init {  atomic{
        run monitor();
        run north_signal();
        run west_signal()
        }
}

/*  [] !(north_aspect == green &&west_aspect  == green)  desired behaviour */
/*                                                                         */
/*  <> (north_aspect == green &&west_aspect  == green)   error behaviour   */
```

✓ ✓ ✓ 3

✓ 1

global system assertion will eventually fail. 1

5

✓ ✓ ✓ ✓ ✓ 5

QUES NO.: Q [redacted] a 2 | EXAM DIET:

2(b)

```
mtype = {green, red};

mtype north_aspect, west_aspect;

proctype north_signal(chan in)
{
        do
        :: atomic{in?red -> north_aspect = red;}
        :: atomic{in?green -> north_aspect = green;}
        od
}                                                        3

proctype west_signal(chan in)
{
        do
        :: atomic{in?green -> west_aspect = green;}
        :: atomic{in?red -> west_aspect = red;}
        od
}                                                        3

proctype control(chan north, west)
{
        do
        ::        north!red; west!green;
        ::        west!red; north!green;

        od
}                                                        6

proctype monitor()
{
        do
        ::assert(!(north_aspect == green && west_aspect  == green))
        od
}

chan north_ctl = [0] of { mtype };
chan west_ctl = [0] of { mtype };

init {  atomic{
        run monitor();
        run control(north_ctl, west_ctl);
        run north_signal(north_ctl);
        run west_signal(west_ctl)
        }
}                                                        3
```