**Exercise 1: (understanding execution of Promela code)**

Consider the following Promela code:

```
proctype A(chan q1)
{ chan q2;
   q1?q2;
   q2!123
}
proctype B(chan qforb)
{ int x;
   qforb?x;
   printf("x=%d\n", x)
}
init {
chan qname = [1] of { chan };
chan qforb = [1] of { int };
run A(qname);
run B(qforb);
qname!qforb
}
```

**Questions:**

1) Propose automata models for this Promela code.
2) Based on the automata models, represent the execution of the system.
3) What is the result of the execution of this code?
4) Make this execution in SPIN

**Exercise 2: (verification with assertion)**

Consider the following Promela code:

```
#define true 1
#define false 0
#define Aturn false
#define Bturn true
bool x, y, t;
proctype A()
   {x = true;
   t = Bturn;
   (y == false || t == Aturn);
   /* critical section */
   x = false
   }
```

```
proctype B()
{y = true;
t = Aturn;
(x == false) ||(t == Bturn);
/* critical section */
y = false
}
```

```
init{ run A(); run B()}
```

**Questions:**

1) Edit this specification in SPIN.
2) Execute this specification
3) Does this specification ensure the *mutual exclusion* in the critical section? (you must think to use an assertion), and if not then show a counter-example.

**Exercise 3: (synchronisation)**
Consider the following promela code

```
#define msgtype 33
     chan name = [0] of { byte, byte };
       proctype A()
       { name!msgtype,124;
         name!msgtype,121;
       }
       proctype B()
       { byte state;
         name?msgtype,state
       }
init{atomic { run A(); run B() }}
```

**Questions:**

1) Can you deduce the value of the variable state at the end of this Promela code? Justify your answer.
2) Is there any code which not reachable? Justify your answer. Verify this using the SPIN. Try to track back the *trail* file (of course, if it is generated by the SPIN).

**Exercise 4:**

Why we need to add sometimes an end label to a code?

2) When we add a progress label, what means this for the SPIN?

**Exercice 5:**

Let see the following promela code :

```
byte x = 0;
byte y = 0;
active proctype A() {
      do
      :: x = 3; progress : skip
      od
}
active proctype B() {
      do
      :: y = 2;
      od
}
```

1) check for progress cycles: with weak fairness ok, without not