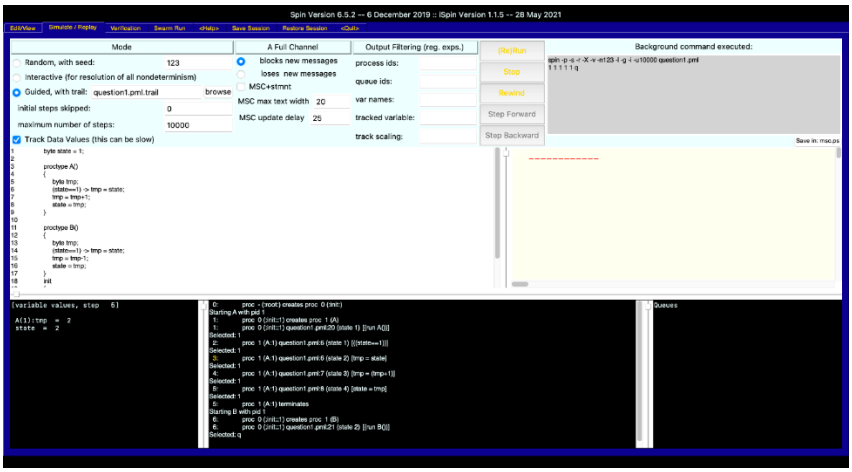
SPIN Modeling Language

Kindly paste the screen shot of both program and verification

1. Consider the following program in promela

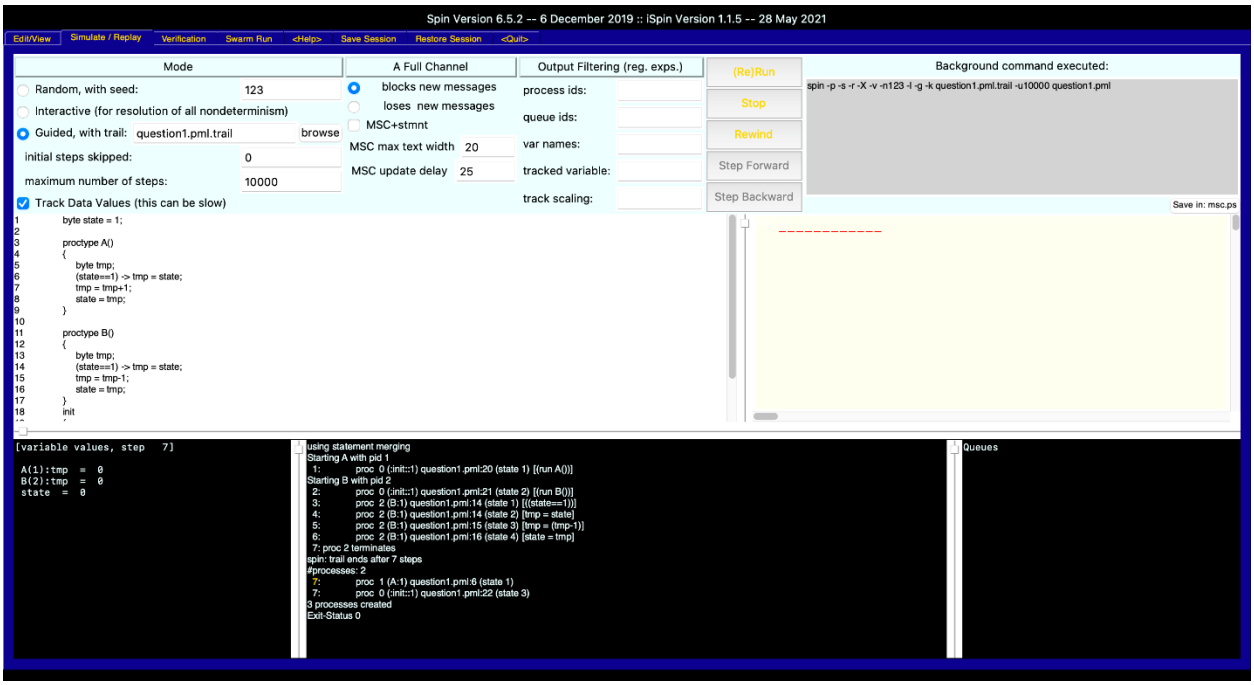
```
byte state = 1;
proctype A()
{
      byte tmp;
      (state==1) -> tmp = state;
      tmp = tmp+1;
      state = tmp;
}
proctype B()
{
     byte tmp;
     (state==1) -> tmp = state;
     tmp = tmp-1;
     state = tmp;
}
init {
    run A();
    run B()
}
```

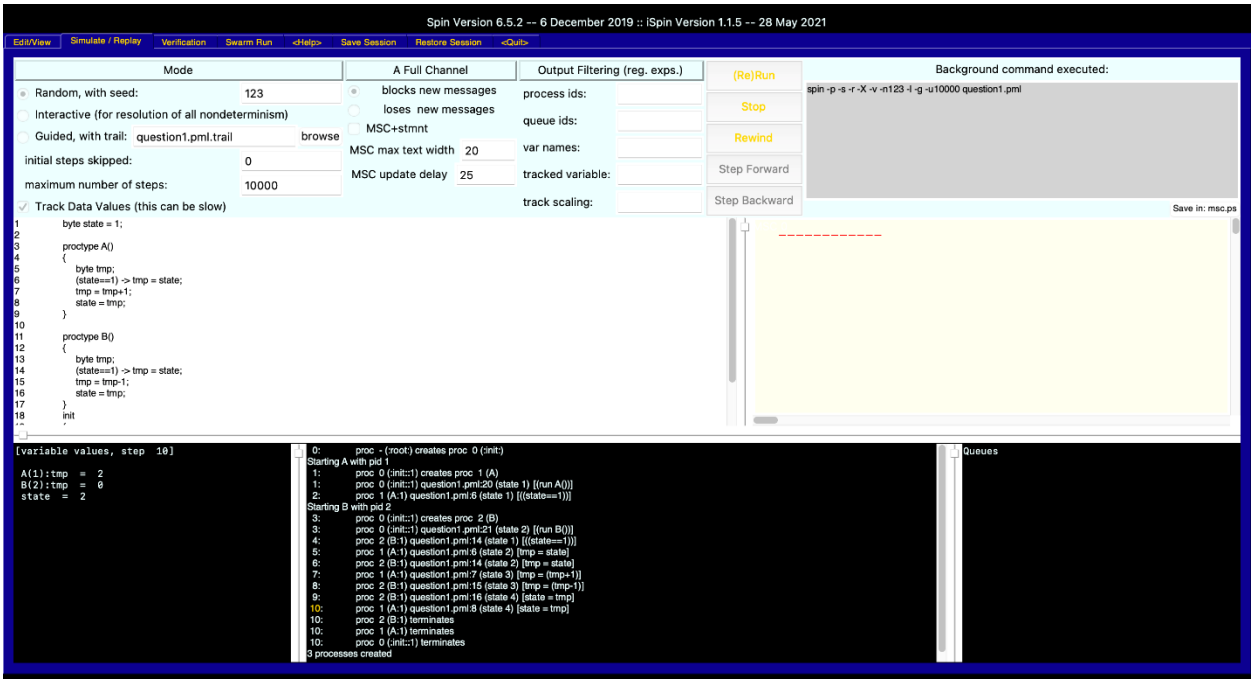
a) Choose the scdehules using ispin option such that process A schdeuled first and terminates followed by process B



b) What will the possible values of state in random simulation.

The possible value of state in random simulation is 0 & 2.





c) change the program to add atomic sequence.

```
byte state = 1;
proctype A()
{
    atomic
    {
        (state == 1) -> state = state+1
    }
}
proctype B()
{
    atomic
    {
        (state == 1) -> state = state-1
    }
}
init
{
    run A();
    run B();
}
```

2. Consider the following program in Promela langauge

```
byte x,t1,t2;
proctype Thread1()
{
    do :: t1 = x;
    t2 = x;
    x = t1 + t2
   od
}
proctype Thread2()
{
    do :: t1 = x;
    t2 = x;
    x = t1 + t2
   od
}
init
{
```

```
    x = 1;
    run Thread1();
    run Thread2();
    assert(x != N)
}
```

for N=1 and N=2 find schedules that cause the assertion to be violated .

At N=1, the schedules that cause assertion violation is
{ x=1 , t1 = 0 , t2 = 0 }
At N=2, the schedules that cause assertion violation is
{ x = 2 , t1 = 1 , t2 = 1 }

```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -p -g question3.pml
  0:    proc  - (:root:) creates proc  0 (:init:)
  1:    proc  0 (:init::1) question3.pml:20 (state 1)   [x = 1]
                x = 1
Starting Thread1 with pid 1
  2:    proc  0 (:init::1) creates proc  1 (Thread1)
  2:    proc  0 (:init::1) question3.pml:21 (state 2)   [(run Thread1())]
Starting Thread2 with pid 2
  3:    proc  0 (:init::1) creates proc  2 (Thread2)
  3:    proc  0 (:init::1) question3.pml:22 (state 3)   [(run Thread2())]
spin: question3.pml:23, Error: assertion violated
spin: text of failed assertion: assert((x!=N))
#processes: 3
                x = 1
                t1 = 0
                t2 = 0
                N = 1
  4:    proc  2 (Thread2:1) question3.pml:12 (state 4)
  4:    proc  1 (Thread1:1) question3.pml:5 (state 4)
  4:    proc  0 (:init::1) question3.pml:23 (state 4)
3 processes created
```

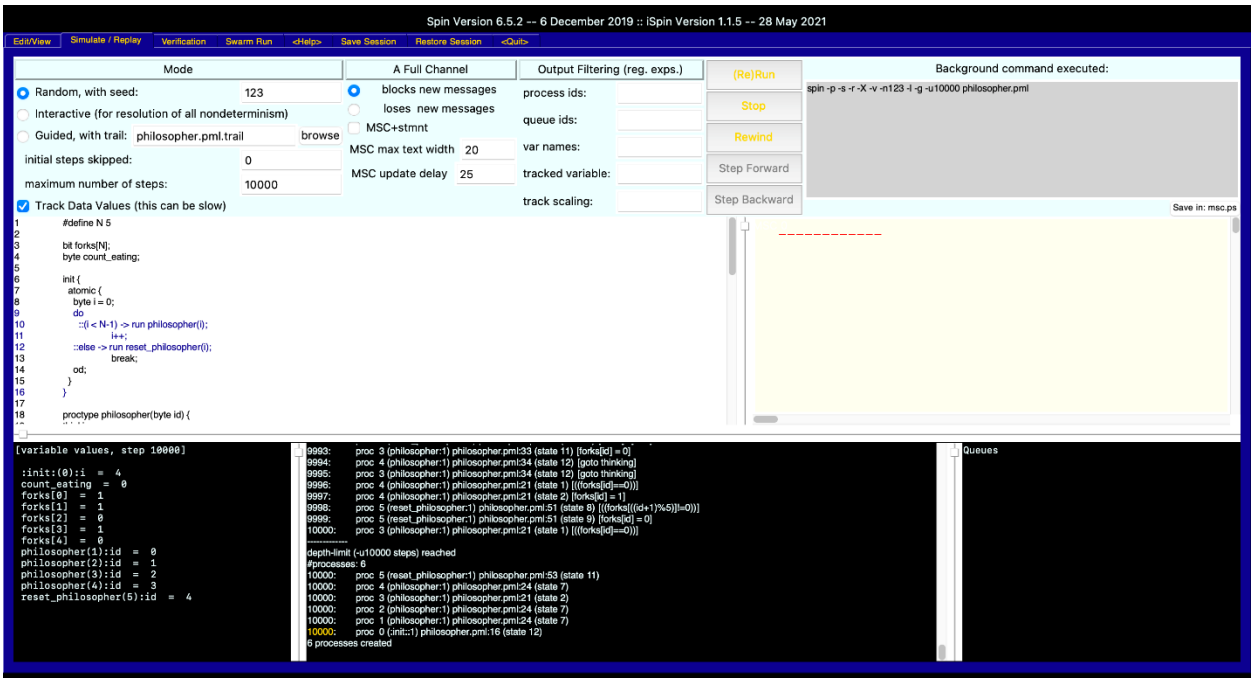
```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -p -g question3.pml
  0:    proc  - (:root:) creates proc  0 (:init:)
  1:    proc  0 (:init::1) question3.pml:20 (state 1)   [x = 1]
                x = 1
Starting Thread1 with pid 1
  2:    proc  0 (:init::1) creates proc  1 (Thread1)
  2:    proc  0 (:init::1) question3.pml:21 (state 2)   [(run Thread1())]
Starting Thread2 with pid 2
  3:    proc  0 (:init::1) creates proc  2 (Thread2)
  3:    proc  0 (:init::1) question3.pml:22 (state 3)   [(run Thread2())]
  4:    proc  1 (Thread1:1) question3.pml:5 (state 1)   [t1 = x]
                t1 = 1
  5:    proc  1 (Thread1:1) question3.pml:6 (state 2)   [t2 = x]
                t2 = 1
  6:    proc  1 (Thread1:1) question3.pml:7 (state 3)   [x = (t1+t2)]
                x = 2
spin: question3.pml:23, Error: assertion violated
spin: text of failed assertion: assert((x!=N))
#processes: 3
                x = 2
                t1 = 1
                t2 = 1
                N = 2
  7:    proc  2 (Thread2:1) question3.pml:12 (state 4)
  7:    proc  1 (Thread1:1) question3.pml:9 (state 5)
  7:    proc  0 (:init::1) question3.pml:23 (state 4)
3 processes created
```

3. Write a promela langauge model for Dining philosopher problem.

```
#define N 5

bit forks[N];
byte count_eating;

init {
  atomic {
    byte i = 0;
    do
      ::(i < N-1) -> run philosopher(i);
        i++;
    ::else -> run reset_philosopher(i);
        break;
    od;
  }
}


proctype philosopher(byte id) {
thinking:
  atomic {
        forks[id] == 0; -> forks[id] = 1;
  }
choosing:
  atomic {
        forks[(id + 1)%N] == 0 -> forks[(id + 1)%N] = 1;
              count_eating++;
  };
eating:
  d_step {
        count_eating--;
        forks[(id + 1)%N] = 0;
  }
  forks[id] = 0;
  goto thinking;
}


proctype reset_philosopher(byte id) {
thinking:
  atomic {
        forks[id] == 0; ->
              forks[id] = 1;
  }
```

```
choosing:
  if
  ::atomic {
          forks[(id + 1)%N] == 0 -> forks[(id + 1)%N] = 1;
                  count_eating++;
      };
  ::atomic {
          forks[(id + 1)%N] != 0 -> forks[id] = 0;
      }
      goto thinking;
  fi;


eating:
  d_step {
      count_eating--;
      forks[(id + 1)%N] = 0;
  }
  forks[id] = 0;
  goto thinking;
}
```

a) For given model using spin do random simulation



b) using spin do verification.

```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -a  philosopher.pml
gcc -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c
./pan -m10000

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (not selected)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 60 byte, depth reached 316, errors: 0
      393 states, stored
     1100 states, matched
     1493 transitions (= stored+matched)
       16 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.033       equivalent memory usage for states (stored*(State-vector + overhead))
    0.283       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage


unreached in init
        (0 of 12 states)
unreached in proctype philosopher
        philosopher.pml:35, state 13, "-end-"
        (1 of 13 states)
unreached in proctype reset_philosopher
        philosopher.pml:63, state 19, "-end-"
        (1 of 19 states)

pan: elapsed time 0 seconds
kumarlaxmikant@Kumars-MacBook-Pro Spin %
```

4. Write a program using PROMELA Language to define following model. Two process do not share any common data.

ProcessA: Computers reverse of a given number 123 digit number.
Process B: Maximum of the given a=3 and b=1.

```
proctype Reverse()
{
   int value=123;
   int reversed=0;
   do
   :: reversed = reversed + (value % 10);
   :: value = value/10;
   :: (value==0) -> break;
   od
   printf("value=%d,reversed=%d\n",value,reversed)
}
proctype Max()
{
    int a=3,b=1;
    int max = (a>=b -> a : b);
```

```
    printf("max=%d\n",max);
}
init
{
   run Reverse();
   run Max();
}
```

```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -p -g first.pml;
  0:    proc  - (:root:) creates proc  0 (:init:)
Starting Reverse with pid 1
  1:    proc  0 (:init::1) creates proc  1 (Reverse)
  1:    proc  0 (:init::1) first.pml:20 (state 1)       [(run Reverse())]
Starting Max with pid 2
  2:    proc  0 (:init::1) creates proc  2 (Max)
  2:    proc  0 (:init::1) first.pml:21 (state 2)       [(run Max())]
  3:    proc  1 (Reverse:1) first.pml:6 (state 1)       [reversed = (reversed+(value%10))]
  4:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
  5:    proc  1 (Reverse:1) first.pml:6 (state 1)       [reversed = (reversed+(value%10))]
            max=3
  6:    proc  2 (Max:1) first.pml:16 (state 1)  [printf('max=%d\\n',max)]
  7:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
  7:    proc  2 (Max:1)               terminates
  8:    proc  1 (Reverse:1) first.pml:6 (state 1)       [reversed = (reversed+(value%10))]
  9:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
 10:    proc  1 (Reverse:1) first.pml:7 (state 2)       [value = (value/10)]
 11:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
 12:    proc  1 (Reverse:1) first.pml:7 (state 2)       [value = (value/10)]
 13:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
 14:    proc  1 (Reverse:1) first.pml:7 (state 2)       [value = (value/10)]
 15:    proc  1 (Reverse:1) first.pml:10 (state 6)      [.(goto)]
 16:    proc  1 (Reverse:1) first.pml:8 (state 3)       [((value==0))]
 17:    proc  1 (Reverse:1) first.pml:8 (state 4)       [goto :b0]
          value=0,reversed=9
 18:    proc  1 (Reverse:1) first.pml:10 (state 8)      [printf('value=%d,reversed=%d\\n',value,reversed)]
 18:    proc  1 (Reverse:1)           terminates
 18:    proc  0 (:init::1)       terminates
3 processes created
kumarlaxmikant@Kumars-MacBook-Pro Spin %
```

5. Write a program using PROMELA Language to define following model. Two process do not share any common data.

ProcessA: Computers whether given number is odd or even.
Process B: Minimum of the given a=3 and b=1.

```
proctype A()
{
    int data = 56;
    if
    :: (data%2==0) -> printf("Data is Even");
    :: else -> printf("Data is Odd");
    fi
}
```

```
proctype B()
{
    int a=3,b=1;
    int min = (a<=b -> a : b);
    printf("min=%d\n",min);
}
init
{
    run A();
    run B();
}
```

```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -p -g second.pml;
  0:    proc  - (:root:) creates proc  0 (:init:)
Starting A with pid 1
  1:    proc  0 (:init::1) creates proc  1 (A)
  1:    proc  0 (:init::1) second.pml:17 (state 1)      [(run A())]
  2:    proc  1 (A:1) second.pml:5 (state 1)    [(((data%2)==0))]
        Data is Even  3:      proc  1 (A:1) second.pml:5 (state 2)    [printf('Data is Even')]
  4:    proc  1 (A:1) second.pml:8 (state 6)    [.(goto)]
  4:    proc  1 (A:1)           terminates
Starting B with pid 1
  5:    proc  0 (:init::1) creates proc  1 (B)
  5:    proc  0 (:init::1) second.pml:18 (state 2)      [(run B())]
        min=1
  6:    proc  1 (B:1) second.pml:13 (state 1)   [printf('min=%d\\n',min)]
  6:    proc  1 (B:1)           terminates
  6:    proc  0 (:init::1)       terminates
3 processes created
kumarlaxmikant@Kumars-MacBook-Pro Spin %
```

6. Write a program using PROMELA Language to define following model. Two process do not share any common data.

ProcessA: Prints odd number from 1 to 50.
Process B: Prints even number from 1 to 50.

```
proctype Odd()
{
    int i=1;
    do
    :: if
      :: (i<50) -> i++
      :: else -> break
      fi
    :: if
      :: (i%2!=0) -> printf("%d\n",i);
      :: else -> skip
      fi
```

```
    od
}
proctype Even()
{
    int i=1;
    do
    :: if
      :: (i<=50) ->i++
      :: else -> break
      fi
    :: if
      :: (i%2==0) -> printf("%d\n",i);
      :: else -> skip
      fi
    od
}
init
{
    run Odd();
    run Even();
}
```

```
720:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
721:    proc  1 (Odd:1) third.pml:6 (state 1)   [((i<50))]
722:    proc  1 (Odd:1) third.pml:6 (state 2)   [i = (i+1)]
723:    proc  1 (Odd:1) third.pml:9 (state 6)   [.(goto)]
724:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
725:    proc  1 (Odd:1) third.pml:11 (state 9)  [else]
726:    proc  1 (Odd:1) third.pml:11 (state 10) [(1)]
727:    proc  1 (Odd:1) third.pml:13 (state 12) [.(goto)]
728:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
729:    proc  1 (Odd:1) third.pml:11 (state 9)  [else]
730:    proc  1 (Odd:1) third.pml:11 (state 10) [(1)]
731:    proc  1 (Odd:1) third.pml:13 (state 12) [.(goto)]
732:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
733:    proc  1 (Odd:1) third.pml:6 (state 1)   [((i<50))]
734:    proc  1 (Odd:1) third.pml:6 (state 2)   [i = (i+1)]
735:    proc  1 (Odd:1) third.pml:9 (state 6)   [.(goto)]
736:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
737:    proc  1 (Odd:1) third.pml:10 (state 7)  [(((i%2)!=0))]
          47
738:    proc  1 (Odd:1) third.pml:10 (state 8)  [printf('%d\\n',i)]
739:    proc  1 (Odd:1) third.pml:13 (state 12) [.(goto)]
740:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
741:    proc  1 (Odd:1) third.pml:6 (state 1)   [((i<50))]
742:    proc  1 (Odd:1) third.pml:6 (state 2)   [i = (i+1)]
743:    proc  1 (Odd:1) third.pml:9 (state 6)   [.(goto)]
744:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
745:    proc  1 (Odd:1) third.pml:6 (state 1)   [((i<50))]
746:    proc  1 (Odd:1) third.pml:6 (state 2)   [i = (i+1)]
747:    proc  1 (Odd:1) third.pml:9 (state 6)   [.(goto)]
748:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
749:    proc  1 (Odd:1) third.pml:6 (state 1)   [((i<50))]
750:    proc  1 (Odd:1) third.pml:6 (state 2)   [i = (i+1)]
751:    proc  1 (Odd:1) third.pml:9 (state 6)   [.(goto)]
752:    proc  1 (Odd:1) third.pml:14 (state 14) [.(goto)]
753:    proc  1 (Odd:1) third.pml:7 (state 3)   [else]
754:    proc  1 (Odd:1) third.pml:7 (state 4)   [goto :b0]
754:    proc  1 (Odd:1)          terminates
754:    proc  0 (:init::1)       terminates
3 processes created
kumarlaxmikant@Kumars-MacBook-Pro Spin %
```

7. Write a program using PROMELA Language to define following model

Process incr: Increments the x value by one everytime if x value is less than 200.
Process decr: Decrements the x value by one everytime if x value is greater than 0.

```
byte x=2;
proctype incr()
{
    atomic
    {
       if
       :: (x<200) -> x++
       fi
    }
}
proctype dec()
{
    atomic
    {
       if
       :: (x>0) -> x--
       fi
    }
}
init
{
    run incr();
    run decr();
}
```

```
kumarlaxmikant@Kumars-MacBook-Pro Spin % spin -p -g fourth.pml;
  0:    proc  - (:root:) creates proc  0 (:init:)
Starting incr with pid 1
  1:    proc  0 (:init::1) creates proc  1 (incr)
  1:    proc  0 (:init::1) fourth.pml:22 (state 1)      [(run incr())]
  2:    proc  1 (incr:1) fourth.pml:7 (state 1) [((x<10))]
  3:    proc  1 (incr:1) fourth.pml:7 (state 2) [x = (x+1)]
                x = 3
  3:    proc  1 (incr:1)           terminates
      timeout
#processes: 1
                x = 3
  3:    proc  0 (:init::1) fourth.pml:23 (state 2)
2 processes created
```