Adishree Annabathula
1001161287

CSE6323 AUTOMATED SOFTWARE ENGINEERING

FINAL EXAM

1. Elevator

   The following are the variables and their types defined for the elevator transition system:

car : {1, 2, 3};	(to record which floor the elevator car is on)
req : {1, 2, 3};	(to record which floor is requested)
carstate : {moving, stop};	(to check the state of the elevator car)
cardoor : {open, closed};	(to check if the elevator car doors are open or closed)
shdoor1 : {open, closed};	(to check if the elevator shaft door on 1^{st} floor is open or closed)
shdoor2 : {open, closed};	(to check if the elevator shaft door on 1^{st} floor is open or closed)
shdoor3 : {open, closed};	(to check if the elevator shaft door on 1^{st} floor is open or closed)

1a) Requirements to formal specifications

(i) □ ((car!=1) -> (shdoor1=closed));
□ ((car!=2) -> (shdoor2=closed));
□ ((car!=3) -> (shdoor3=closed));

(ii) □ ((car=1) -> ◊ (shdoor1=open & cardoor=open));
□ ((car=2) -> ◊ (shdoor2=open & cardoor=open));
□ ((car=3) -> ◊ (shdoor3=open & cardoor=open));

Clarification:
□ ((car=1 & req=1) -> ◊ (shdoor1=open & cardoor=open));
□ ((car=2 & req=2) -> ◊ (shdoor2=open & cardoor=open));
□ ((car=3 & req=3) -> ◊ (shdoor3=open & cardoor=open));

(iii) □ ((carstate=moving) -> (shdoor1=closed & shdoor2=closed & shdoor3=closed));

(iv) □ ((req=1) -> ◊ (car=1));
□ ((req=2) -> ◊ (car=2));
□ ((req=3) -> ◊ (car=3));

(v) □ ((cardoor=closed) <-> (shdoor1=closed & shdoor2=closed & shdoor3=closed));

(vi) (shdoor1=open) -> (shdoor2=closed & shdoor3=closed);
(shdoor2=open) -> (shdoor3=closed & shdoor1=closed);
(shdoor3=open) -> (shdoor1=closed & shdoor2=closed);

Safety requirement – (i) Liveness requirements – (ii), (iv)

1b-e are available in the model file named elevator.smv.

```
##############################################################
The transition relation is not total. A state without
successors is:
car = 2
req = 1
carstate = stop
cardoor = closed
shdoor1 = closed
shdoor2 = closed
shdoor3 = closed
However, all the states without successors are
non-reachable, so the machine is deadlock-free.
##############################################################
-- specification  G (car != 1 -> shdoor1 = closed)  is true
-- specification  G (car != 2 -> shdoor2 = closed)  is true
-- specification  G (car != 3 -> shdoor3 = closed)  is true
-- specification  G (car = 1 ->  F (shdoor1 = open & cardoor = open))  is true
-- specification  G (car = 2 ->  F (shdoor2 = open & cardoor = open))  is true
-- specification  G (car = 3 ->  F (shdoor3 = open & cardoor = open))  is true
-- specification  G ((car = 1 & req = 1) ->  F (shdoor1 = open & cardoor = open))  is true
-- specification  G ((car = 2 & req = 2) ->  F (shdoor2 = open & cardoor = open))  is true
-- specification  G ((car = 3 & req = 3) ->  F (shdoor3 = open & cardoor = open))  is true
-- specification  G (carstate = moving -> ((shdoor1 = closed & shdoor2 = closed) & shdoor3 = closed))  is true
-- specification  G (req = 1 ->  F car = 1)  is true
-- specification  G (req = 2 ->  F car = 2)  is true
-- specification  G (req = 3 ->  F car = 3)  is true
-- specification  G (cardoor = closed <-> ((shdoor1 = closed & shdoor2 = closed) & shdoor3 = closed))  is true
-- specification (shdoor1 = open -> (shdoor2 = closed & shdoor3 = closed))  is true
-- specification (shdoor2 = open -> (shdoor3 = closed & shdoor1 = closed))  is true
-- specification (shdoor3 = open -> (shdoor1 = closed & shdoor2 = closed))  is true
```

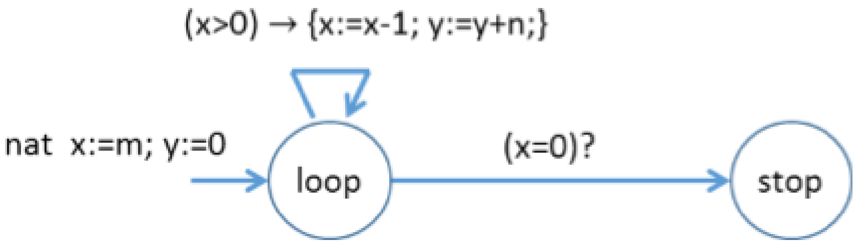
1b) LTLSPEC
1c) State variables and their types
1d) Initial state:

(shdoor1=closed & shdoor2=closed & shdoor3=closed & cardoor=closed & carstate=stop & car=1);

1e) One way of checking the LTL specifications of a transition system is by using the command check_ltlspec in the interactive mode.
Use the command
nuxmv.exe –int mult.smv

```
nuXmv > go
nuXmv > check_ltlspec
-- specification  G (car != 1 -> shdoor1 = closed)  is true
-- specification  G (car != 2 -> shdoor2 = closed)  is true
-- specification  G (car != 3 -> shdoor3 = closed)  is true
-- specification  G (car = 1 ->  F (shdoor1 = open & cardoor = open))  is true
-- specification  G (car = 2 ->  F (shdoor2 = open & cardoor = open))  is true
-- specification  G (car = 3 ->  F (shdoor3 = open & cardoor = open))  is true
-- specification  G ((car = 1 & req = 1) ->  F (shdoor1 = open & cardoor = open))  is true
-- specification  G ((car = 2 & req = 2) ->  F (shdoor2 = open & cardoor = open))  is true
-- specification  G ((car = 3 & req = 3) ->  F (shdoor3 = open & cardoor = open))  is true
-- specification  G (carstate = moving -> ((shdoor1 = closed & shdoor2 = closed) & shdoor3 = closed))  is true
-- specification  G (req = 1 ->  F car = 1)  is true
-- specification  G (req = 2 ->  F car = 2)  is true
-- specification  G (req = 3 ->  F car = 3)  is true
-- specification  G (cardoor = closed <-> ((shdoor1 = closed & shdoor2 = closed) & shdoor3 = closed))  is true
-- specification (shdoor1 = open -> (shdoor2 = closed & shdoor3 = closed))  is true
-- specification (shdoor2 = open -> (shdoor3 = closed & shdoor1 = closed))  is true
-- specification (shdoor3 = open -> (shdoor1 = closed & shdoor2 = closed))  is true
nuXmv > quit
```

2. Mult(m,n) transition system



The transition system T = <S, Init, Trans>

where S is a set if state variables

Qs = {x, y, mode}

where mode ranges over {loop, stop}

Init is an initialization code fragment

Init] = {x:=m; y:=0; mode:=loop}

Trans is a code fragment to describe how the system evolves through the set of transitions between the states

(loop, k) -> (loop, k)

(loop, k) -> (stop, 0)

2a) A property ψ of a transition system T is an inductive invariant of T if:

- Every initial state satisifies ψ.
- If a state s satisfies ψ and (s,t) is a transition, then the state t also satisifies ψ.

Given invariant ψ ≡ (mode=stop) -> (y=m*n)
Let m=3 and n=1, i.e., Mult(m,n)=Mult(3,1)
Base case: Substitute the initial state in the given invariant.
Initial state is {x:=m; y:=0; mode:=loop} = { x:=3; y:=0; mode:=loop}
ψ ≡ (loop=stop) -> (0=3*1) ≡ F -> F ≡ ~F V F ≡ T
So the base case holds true for ψ.
Inductive case:
Assume state s={x:=2; y:=1; mode:=loop} which holds true for ψ and (s,t) is a transition.
So state t={x:=1; y:=2; mode:=stop}
ψ≡ (stop=stop) -> (2=3*1) ≡ T -> F ≡ ~T V F ≡ F
State t doesn't hold true for ψ.

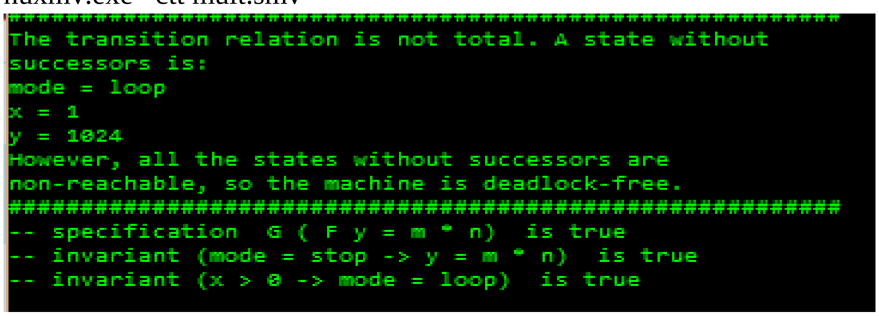
Since the induction case fails, we can conclude that the given invariant ψ ≡ (mode=stop) -> (y=m*n) is not an inductive invariant.

2b) Consider an invariant φ ≡ (x>0) -> (mode=loop)
Let m=3 and n=1, i.e., Mult(m,n)=Mult(3,1)
Base case: Substitute the initial state in the given invariant.
Initial state is {x:=m; y:=0; mode:=loop} = { x:=3; y:=0; mode:=loop}
φ ≡ (3>0) -> (loop=loop) ≡ T -> T ≡ ~T V T ≡ T
So the base case holds true for φ.

Inductive case:
Assume state s={x:=2; y:=1; mode:=loop} which holds true for ψ and (s,t) is a transition.
So state t={x:=1; y:=2; mode:=loop}
$\varphi \equiv (1>0) \to (loop=loop) \equiv T \to T \equiv \sim T \vee T \equiv T$
State t holds true for φ.
We can conclude that the invariant $\varphi \equiv (x>0) \to (mode=loop)$ is an inductive invariant.

A property P is stronger than a property Q iff P->Q.
Here $P \equiv \psi \equiv (mode=stop) \to (y=m*n)$ and $Q \equiv \varphi \equiv (x>0) \to (mode=loop)$
Base case: Substitute the initial state in P->Q.
Initial state is {x:=m; y:=0; mode:=loop} = { x:=3; y:=0; mode:=loop}
$\varphi \equiv T \to T \equiv \sim T \vee T \equiv T$
So the base case holds true.
Inductive case:
Assume state s={x:=2; y:=1; mode:=loop} which holds true for ψ and (s,t) is a transition.
So state t={x:=1; y:=2; mode:=loop}
$\varphi \equiv F \to T \equiv \sim F \vee T \equiv T$
State t holds true.
We can conclude that P is a stronger property than Q.

2c) The model file named mult.smv contains the code.
The transition system is not total and is deadlock free as shown in the screenshot in 2d.

2d) Use the command
nuxmv.exe –ctt mult.smv

```
######################################################################
The transition relation is not total. A state without
successors is:
mode = loop
x = 1
y = 1024
However, all the states without successors are
non-reachable, so the machine is deadlock-free.
######################################################################
-- specification  G ( F y = m * n)  is true
-- invariant (mode = stop -> y = m * n)  is true
-- invariant (x > 0 -> mode = loop)  is true
```
The invariant from 2a is indeed an invariant.

2e) Use the command

nuxmv.exe –bmc mult.smv

```
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- no counterexample found with bound 5
-- no counterexample found with bound 6
-- no counterexample found with bound 7
-- no counterexample found with bound 8
-- no counterexample found with bound 9
-- no counterexample found with bound 10
-- cannot prove the invariant (mode = stop -> y = m * n)  is true or false : the induction fails
-- as demonstrated by the following execution sequence
Trace Description: BMC Failed Induction
Trace Type: Counterexample
  -> State: 1.1 <-
    mode = loop
    x = 0
    y = 0
    m = 3
    n = 1
  -> State: 1.2 <-
    mode = stop
-- invariant (x > 0 -> mode = loop)   is true
```

The invariant from 2b is an inductive invariant.
Using the bmc option, we observe that induction fails for the invariant from 2a.

2f) **LTL liveness requirement**: It is always the case that eventually a state where y is m*n is reached.

One way of checking the LTL specifications of a transition system is by using the command check_ltlspec in the interactive mode.

Use the command

nuxmv.exe –int mult.smv

```
nuXmv > go
nuXmv > check_ltlspec
-- specification  G ( F y = m * n)  is true
nuXmv > quit
```