Formal Methods

Assignment #3

Mehmoona Bibi
#111299
BESE 5B

Q1: *Prepare and Enhance in the example of Light with various levels of brightness.*

Light model:

Light model has three levels (light, medium, bright). I have also included mutual exclusion principle in that is defined below.

Mutual exclusion:

Mutual exclusion model is a process which prevents two processes to access shared resources. The concept is used in concurrent programming which has a critical section in which there are shared resources.

Locations of Light1, Light2:

Off, light, medium, bright

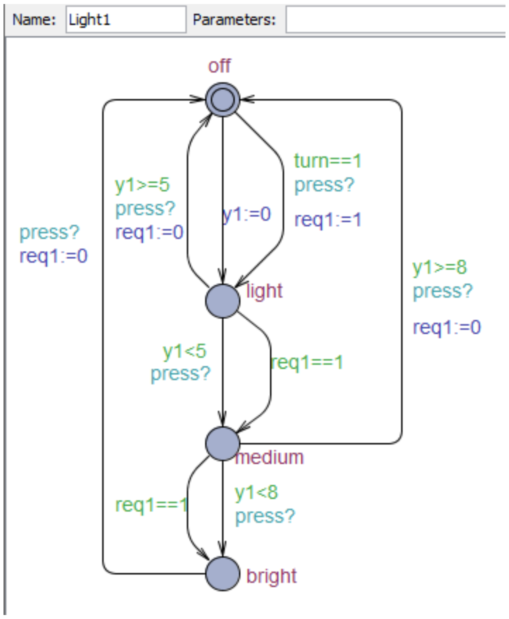
Locations of User:

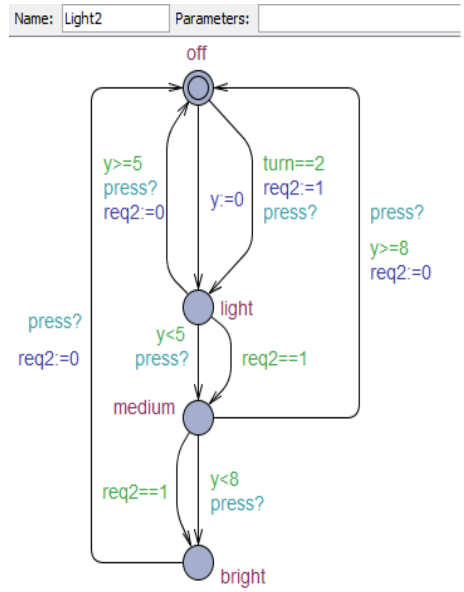
idle

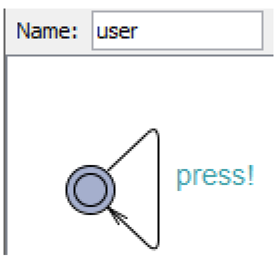
Process 1:	**Process 2:**	**User:**

off:
req1=1;
while(turn!=1);
press?
Y1==0;
light:
while(req1!=1 && y1<5);
press?
medium:
while(req1!=1 && y1<8);
press?
bright:
press?
req1=0;
//and return to off
	off:
	req2=1;
	while(turn!=2);
	press?
	y==0;
	light:
	while(req2!=1 && y<5);
	press?
	medium:
	while(req2!=1 && y<8);
	press?
	bright:
	press?
	req2=0;
	//and return to off and return to idle
		Idle:
		press!

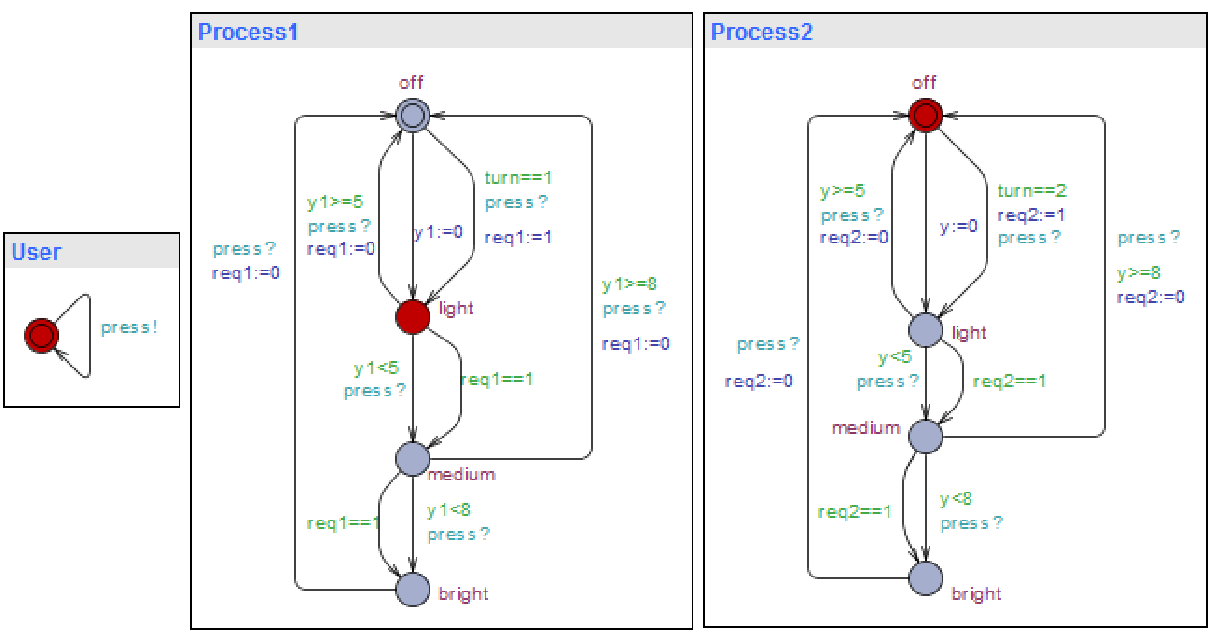
Automaton:



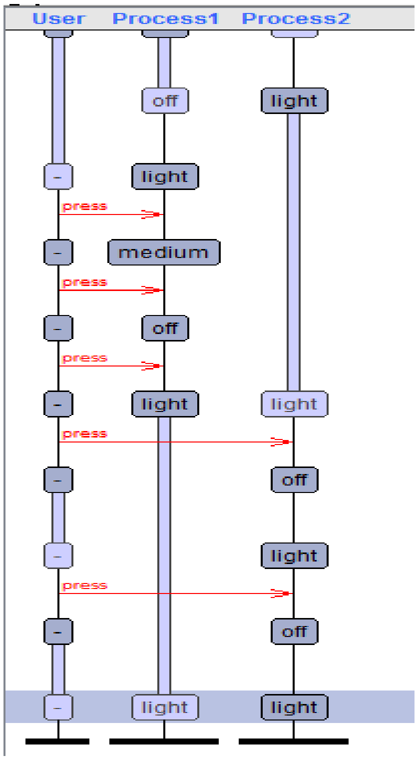




Simulation:



Simulation Trace:



Simulation Trace

(-, bright, off)
press: User → Process1
(-, off, off)
Process2
(-, off, light)
press: User → Process2
(-, off, off)
Process2
(-, off, light)
press: User → Process2
(-, off, off)
press: User → Process1
(-, light, off)

System declarations:

```
// Place template instantiations here.
User = user();
Process1 = Light1();
Process2 = Light2();
// List one or more processes to be composed into a system.
system User,Process1,Process2;
```

Declarations:

```
// Place global declarations here.
chan press;
int [1,2] turn;
int [0,1] req1,req2;
```

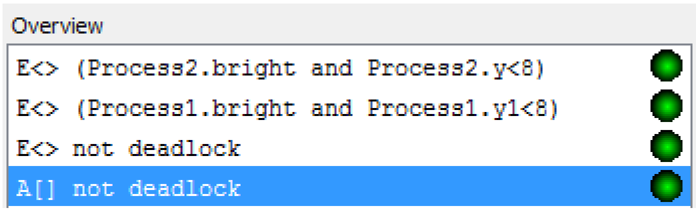
Name: Light1

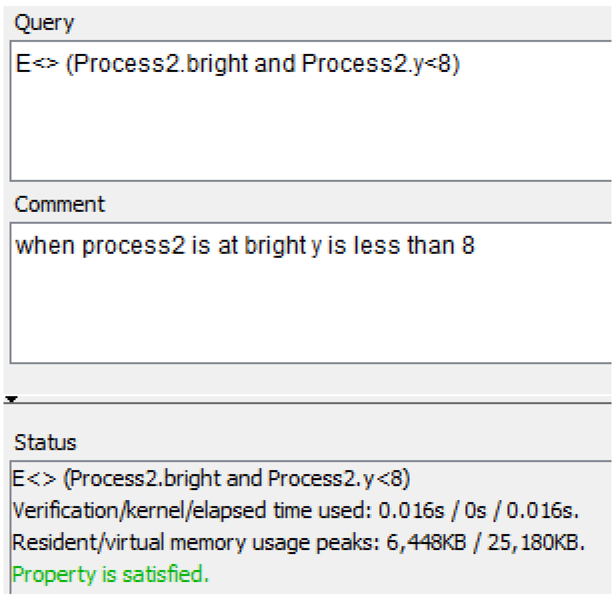
```
clock y1;
```

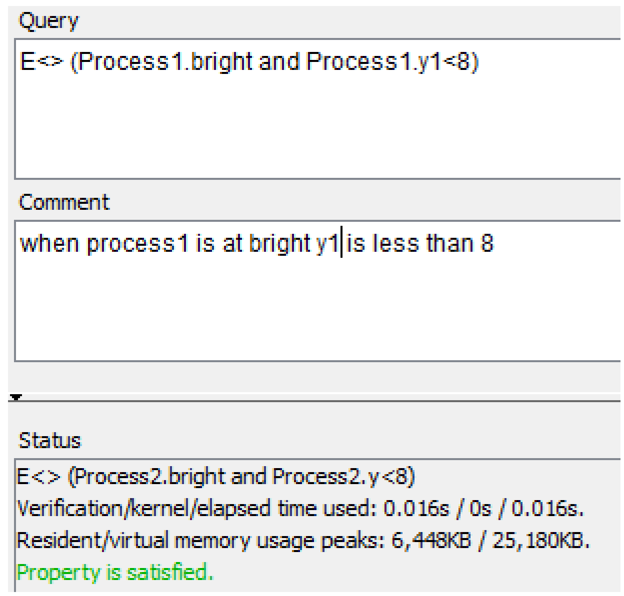
Name: Light2

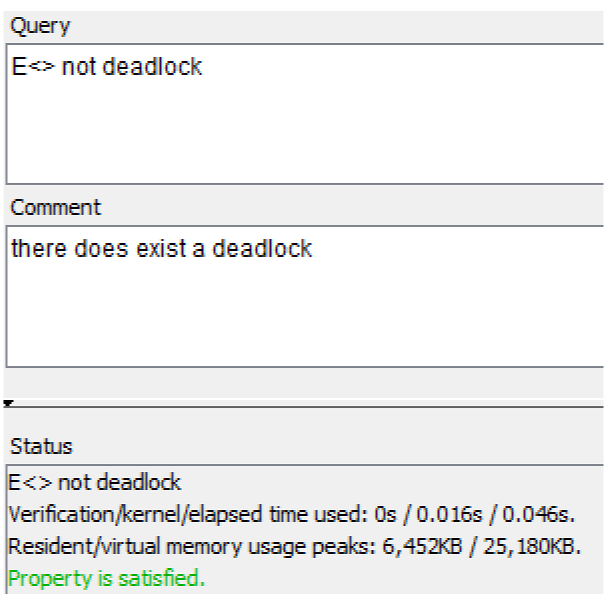
```
clock y;
```

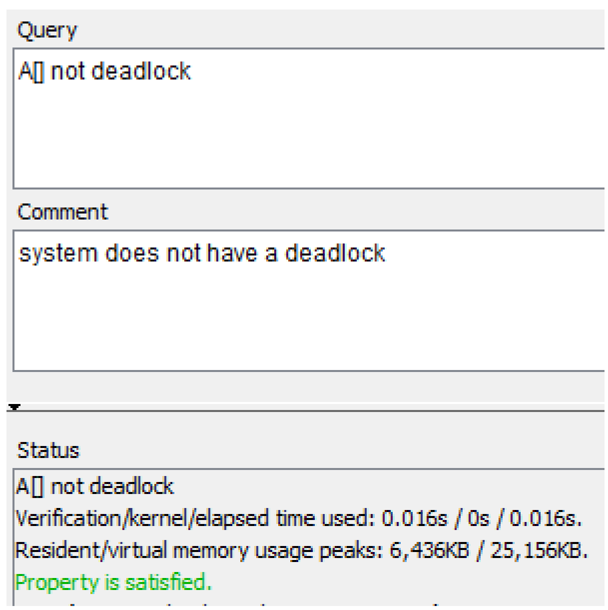
Verification:











Q2: Write an example of your own.

Train gate example:

Train gate automaton is a railway control automaton in which 6 trains pass through a single gate and no one collides. There are timing constraints for trains before entering the bridge. When approaching a train sends a appr! signal. Thereafter, it has 10 time units to receive a stop signal. This allows it to stop safely before the bridge. After these 10 time units, it takes further 10 time units to reach the bridge if the train is not stopped. If a train is stopped, it resumes its course when the controller sends a go! signal to it after a previous train has left the bridge and sent a leave! signal.

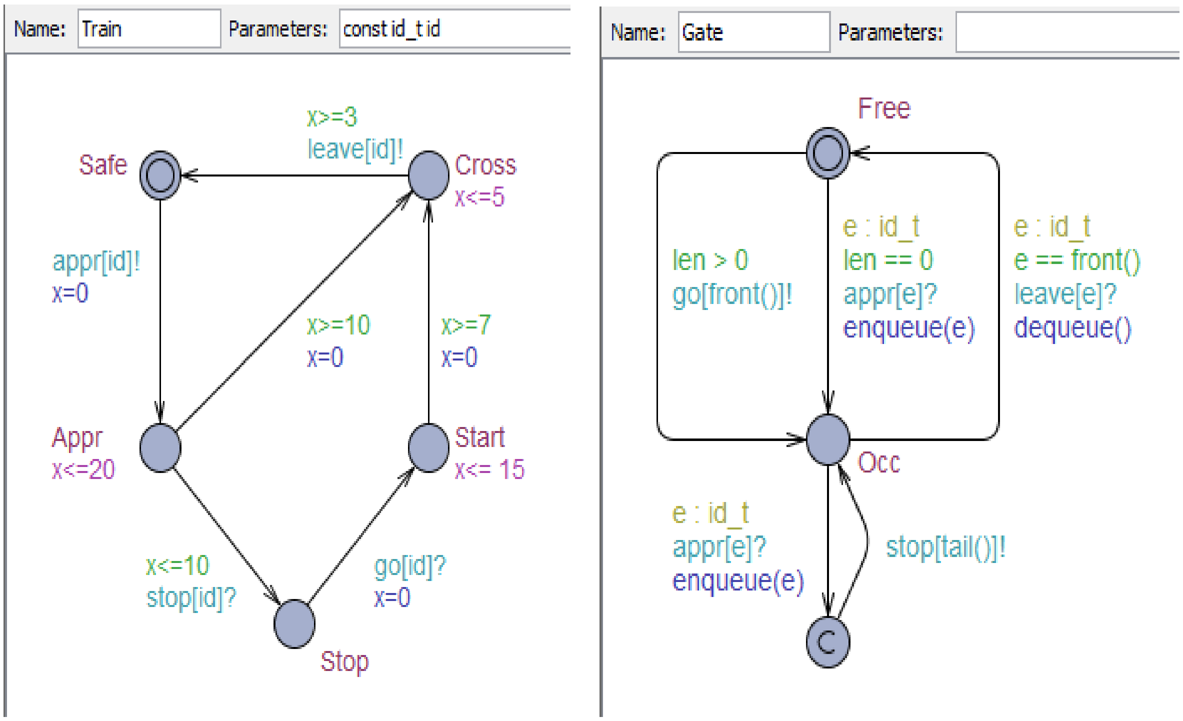
Train	Gate

safe:
appr[id]!
x=0
cross:
while(x>=3)
leave[id]!
start:
while(x>=7);
x=0
Appr:
x>=10
x=0
Stop:
go[id]?
x=0
	free:
	appr[e]?
	len==0
	enqueue(e)
	Occ:
	appr[e]?
	e:id_t
	enqueue(e)
	C:
	stop[tail()]!

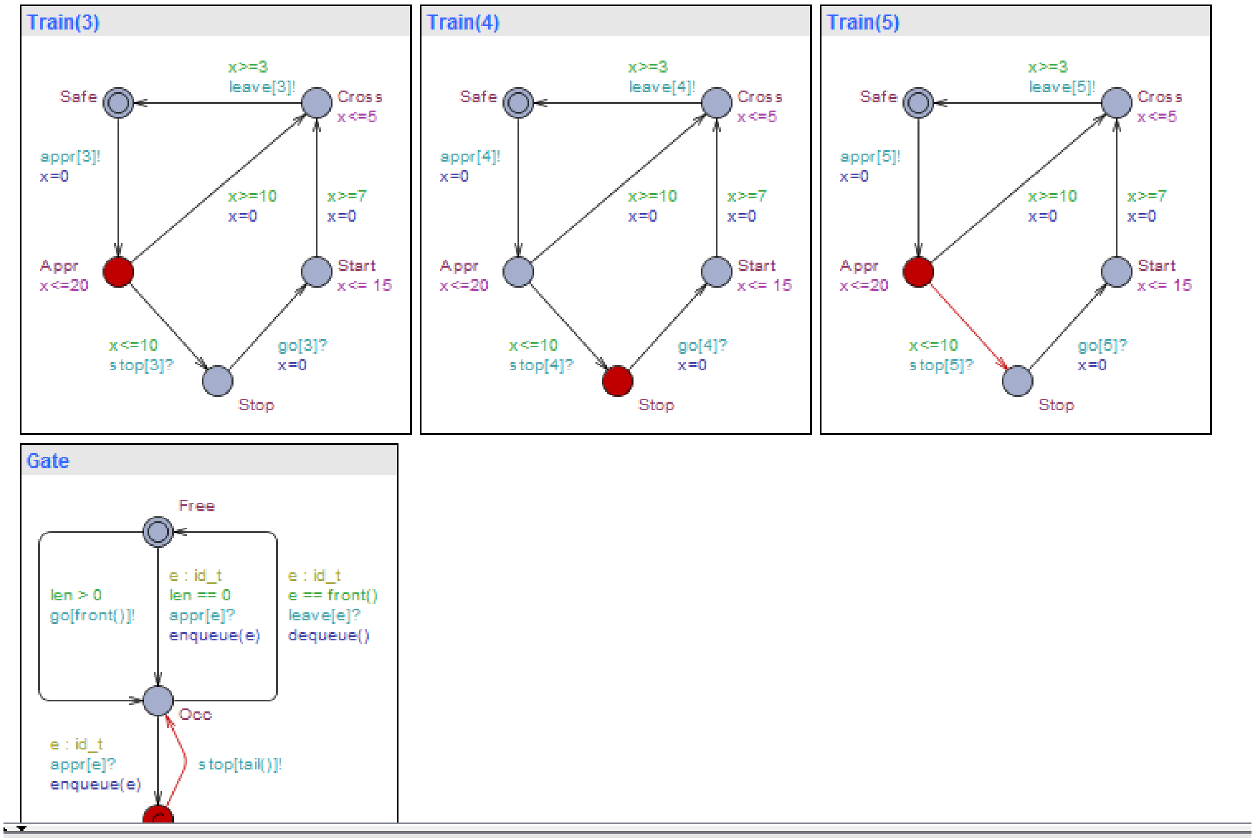
There are two templates:

1. **Train :**
    a. **Locations:**
        i. Safe
        ii. Cross
        iii. Start
        iv. Appr
        v. Stop
2. **Gate**
    a. **Locations:**
        i. Free
        ii. Occ
        iii. C

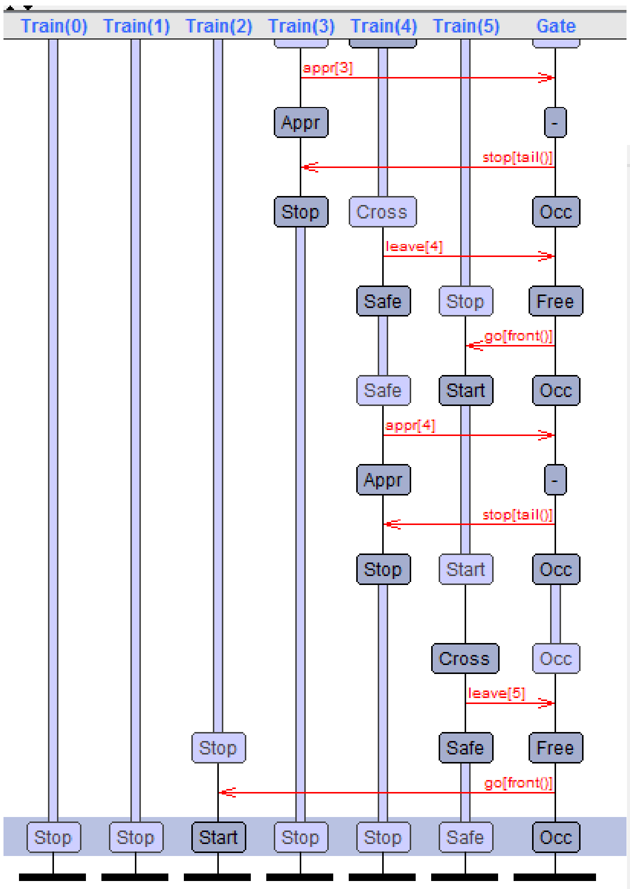
Automaton:

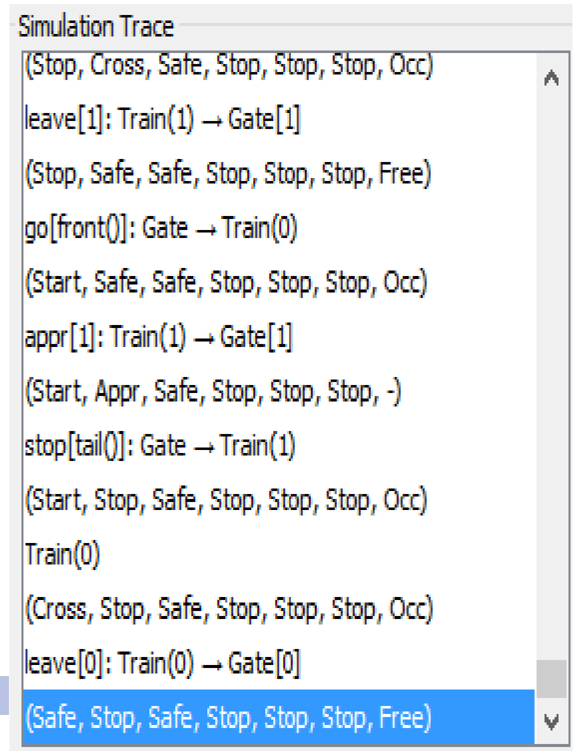

Name: Train
Parameters: const id_t id
Safe
x>=3
leave[id]!
Cross
x<=5
appr[id]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[id]?
go[id]?
x=0
Stop
Name: Gate
Parameters:
Free
len > 0
go[front()]!
e : id_t
len == 0
appr[e]?
enqueue(e)
e : id_t
e == front()
leave[e]?
dequeue()
Occ
e : id_t
appr[e]?
enqueue(e)
stop[tail()]!
C

Simulation:


Train(3)
Safe
Cross
x<=5
x>=3
leave[3]!
appr[3]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[3]?
go[3]?
x=0
Stop
Train(4)
Safe
Cross
x<=5
x>=3
leave[4]!
appr[4]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[4]?
go[4]?
x=0
Stop
Train(5)
Safe
Cross
x<=5
x>=3
leave[5]!
appr[5]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[5]?
go[5]?
x=0
Stop
Gate
Free
len > 0
go[front()]!
e : id_t
len == 0
appr[e]?
enqueue(e)
e : id_t
e == front()
leave[e]?
dequeue()
Occ
e : id_t
appr[e]?
enqueue(e)
stop[tail()]!

Simulation trace:





System declarations:

```
system Train, Gate;
```

Declarations:

```
const int N = 6;          // # trains
typedef int[0,N-1] id_t;

chan        appr[N], stop[N], leave[N];
urgent chan go[N];
```

Name: Train Parameters: const id_t id

```
clock x;
```

Name: Gate Parameters:

```
id_t list[N+1];
int[0,N] len;
// Put an element at the end of the queue
void enqueue(id_t element)
{
        list[len++] = element;
}
// Remove the front element of the queue
void dequeue()
{
        int i = 0;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
}
// Returns the front element of the queue
id_t front()
{
   return list[0];
}
// Returns the last element of the queue
id_t tail()
{
   return list[len - 1];
}
```

Verification:

Overview

```
E<> Gate.Occ
E<> Train(0).Cross
E<> Train(1).Cross
E<> Train(0).Cross and Train(1).Stop
E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)
A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j
A[] Gate.list[N] == 0
Train(0).Appr --> Train(0).Cross
Train(1).Appr --> Train(1).Cross
Train(2).Appr --> Train(2).Cross
Train(3).Appr --> Train(3).Cross
Train(4).Appr --> Train(4).Cross
Train(5).Appr --> Train(5).Cross
A[] not deadlock
```

Query

E<> Gate.Occ

Comment

Gate can receive (and store in queue) msg's from approaching trains.

Status

E<> Gate.Occ
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 7,872KB / 27,924KB.
Property is satisfied.

Query

E<> Train(0).Cross

Comment

Train 0 can reach crossing.

Status

E<> Train(0).Cross
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 7,300KB / 26,928KB.
Property is satisfied.

Query

E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)

Comment

Train 0 can cross bridge while the other trains are waiting to cross.

Status

E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)
Verification/kernel/elapsed time used: 0.218s / 0s / 0.21s.
Resident/virtual memory usage peaks: 7,620KB / 27,516KB.
Property is satisfied.

Query

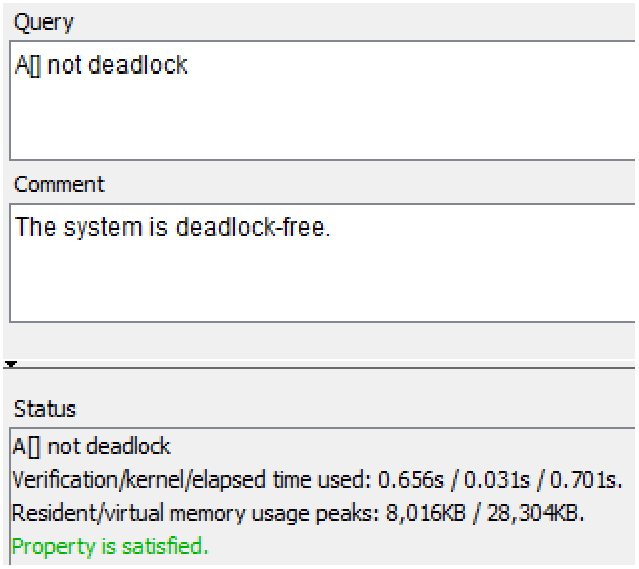
A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j

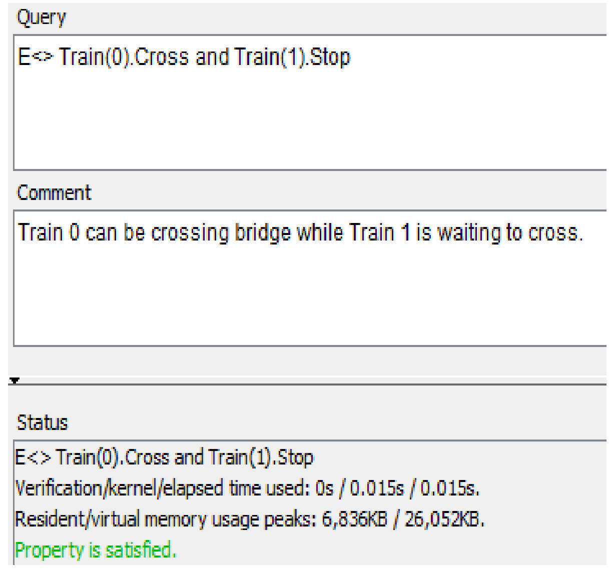
Comment

There is never more than one train crossing the bridge (at any time instance).

Status

A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j
Verification/kernel/elapsed time used: 0.672s / 0.016s / 0.678s.
Resident/virtual memory usage peaks: 8,264KB / 29,108KB.
Property is satisfied.





Github Link:

https://github.com/Mehmoona-bibi/Automaton/