Railway Crossing Gate Controller

Design and Model Verification

GROUP 5:

SAHIB KHAN (160101057)
SAURABH BAZARI (160101061)

Problem Description

Consider a railway track crossing, that is a road crosses a railway line.

We have to design a railway crossing gate which is built on the road to prevent vehicles from crossing while a train is passing.

Assumption

1. The train arrives non deterministically on the track.
2. Before the train arrives it sounds a Horn. Assume that the horn is sounded when the train is sufficiently far for the gate to close.
3. Only one train arrives at a time.
4. We can’t control the train if it arrives it will non-stop.
5. The vehicle will stop when the horn is sounded.

Modules

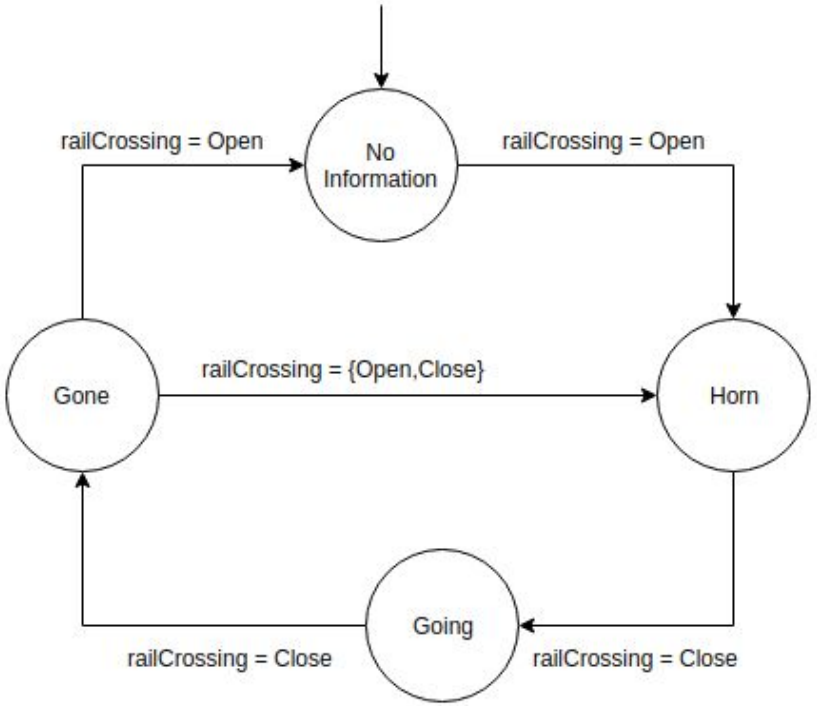
1. Train Module
2. Railway Crossing Gate Module
3. Vehicle Module
4. Main Module

Train Module

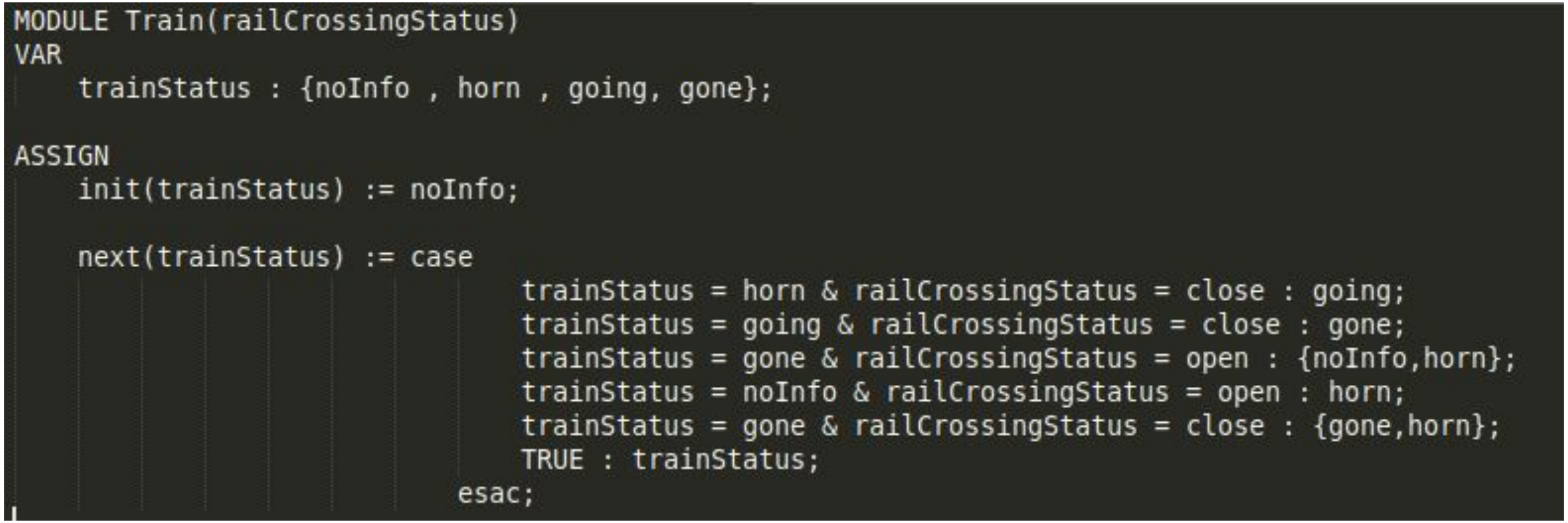
This module stores the status of the train.

A variable trainStatus is maintained which can take one of the 4 values:

1. NoInfo
2. Horn
3. Going
4. Gone



Train Module



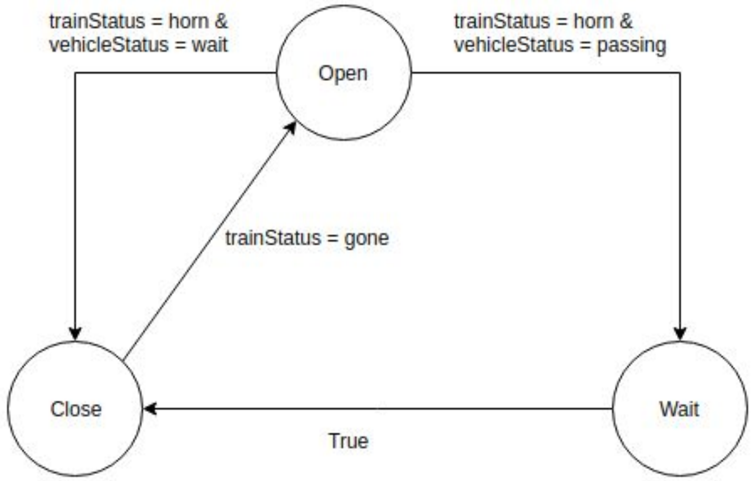
```
MODULE Train(railCrossingStatus)
VAR
    trainStatus : {noInfo , horn , going, gone};

ASSIGN
    init(trainStatus) := noInfo;

    next(trainStatus) := case
                                trainStatus = horn & railCrossingStatus = close : going;
                                trainStatus = going & railCrossingStatus = close : gone;
                                trainStatus = gone & railCrossingStatus = open : {noInfo,horn};
                                trainStatus = noInfo & railCrossingStatus = open : horn;
                                trainStatus = gone & railCrossingStatus = close : {gone,horn};
                                TRUE : trainStatus;
                            esac;
```

RAILWAY CROSSING GATE MODULE

This module stores the state of the crossing gate.

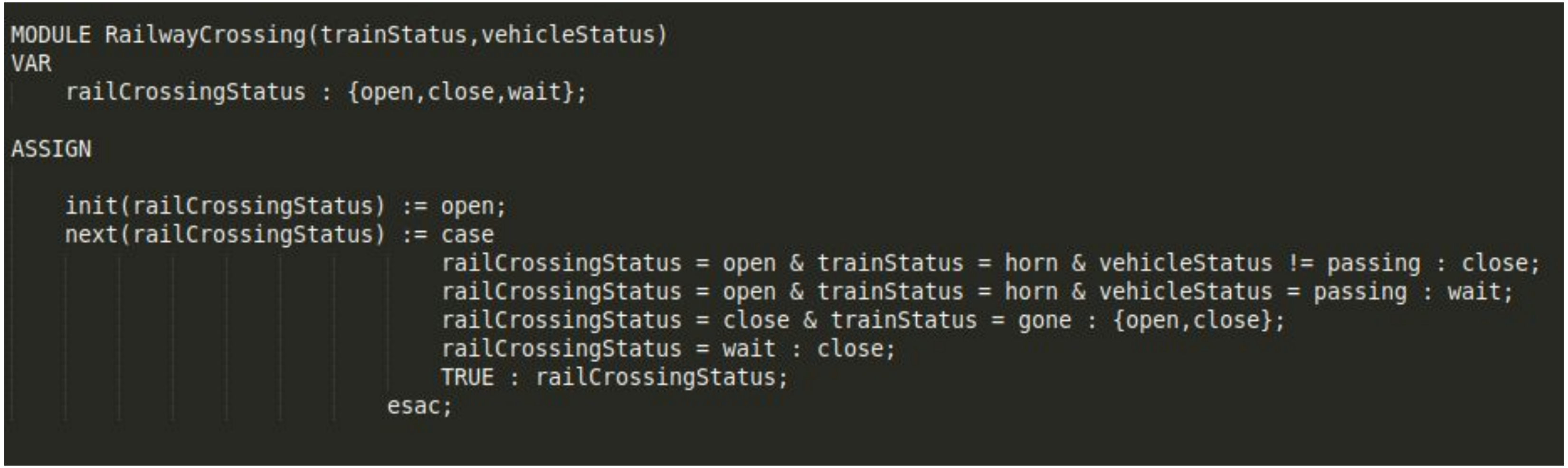
The gate state can be:

- Closed
- Open
- Wait



trainStatus = horn & vehicleStatus = wait
Open
trainStatus = horn & vehicleStatus = passing
trainStatus = gone
Close
Wait
True

RAILWAY CROSSING GATE MODULE



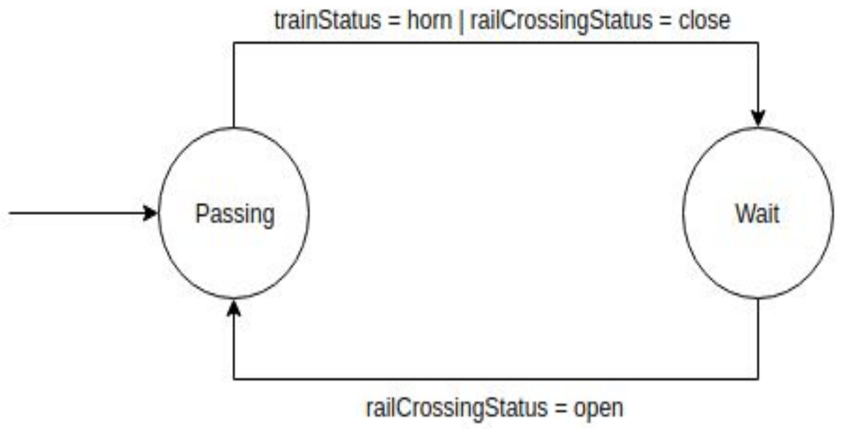
```
MODULE RailwayCrossing(trainStatus,vehicleStatus)
VAR
    railCrossingStatus : {open,close,wait};

ASSIGN

    init(railCrossingStatus) := open;
    next(railCrossingStatus) := case
                                    railCrossingStatus = open & trainStatus = horn & vehicleStatus != passing : close;
                                    railCrossingStatus = open & trainStatus = horn & vehicleStatus = passing : wait;
                                    railCrossingStatus = close & trainStatus = gone : {open,close};
                                    railCrossingStatus = wait : close;
                                    TRUE : railCrossingStatus;
                                esac;
```

Vehicle Module

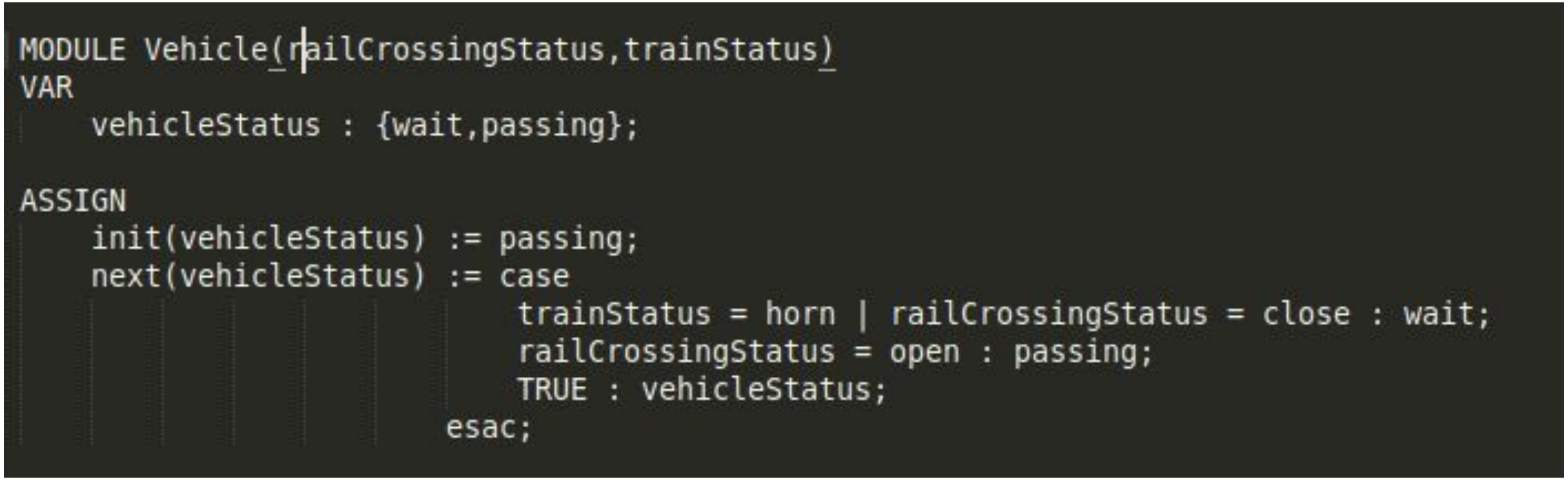
This module is meant for the vehicles which need to pass the crossing.

The vehicleStatus state can be:

- Passing
- Wait


trainStatus = horn | railCrossingStatus = close
Passing
Wait
railCrossingStatus = open

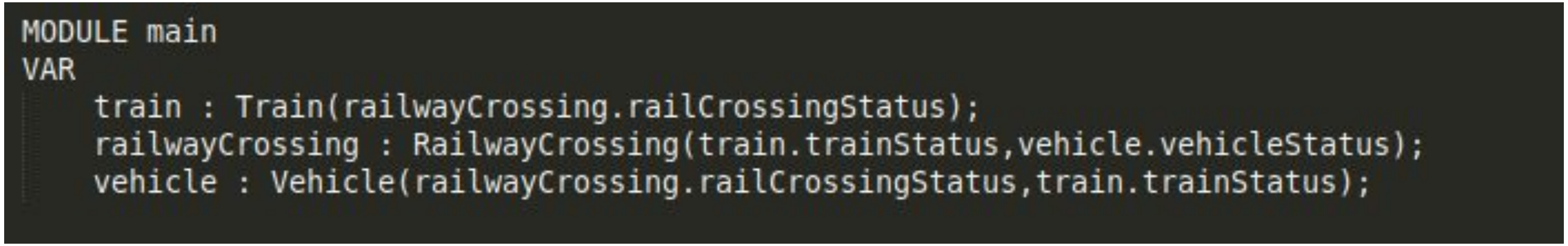
Vehicle Module



```
MODULE Vehicle(railCrossingStatus,trainStatus)
VAR
    vehicleStatus : {wait,passing};

ASSIGN
    init(vehicleStatus) := passing;
    next(vehicleStatus) := case
                                trainStatus = horn | railCrossingStatus = close : wait;
                                railCrossingStatus = open : passing;
                                TRUE : vehicleStatus;
                            esac;
```

Main Module

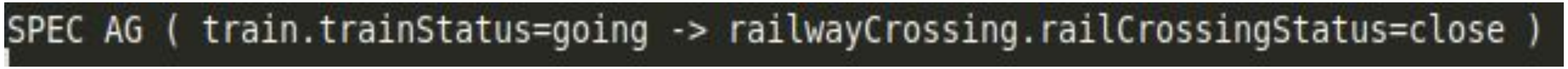
This module is responsible for making instances of the above modules and passing the parameters required by each other.



```
MODULE main
VAR
    train : Train(railwayCrossing.railCrossingStatus);
    railwayCrossing : RailwayCrossing(train.trainStatus,vehicle.vehicleStatus);
    vehicle : Vehicle(railwayCrossing.railCrossingStatus,train.trainStatus);
```

Requirements

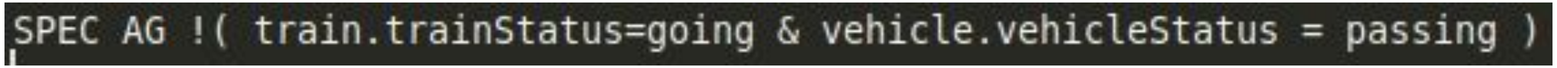
Safety: This is the most important requirement among others. Because we must ensure that the gate is closed while the train is passing because otherwise, accidents may happen. In the below spec we ensure that while a train is passing the gate should be closed to prevent cars from getting near the train.



```
SPEC AG ( train.trainStatus=going -> railwayCrossing.railCrossingStatus=close )
```

Requirements

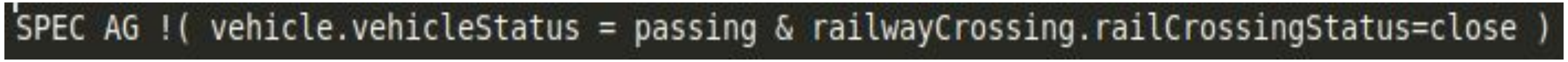
Safety : We are ensuring that a vehicle can’t be passing and the train can’t be going simultaneously. If this condition is violated then the code is not safe.



```
SPEC AG !( train.trainStatus=going & vehicle.vehicleStatus = passing )
```

Requirements

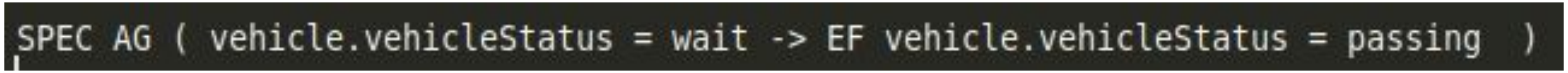
We must ensure that while the vehicle is passing the gate must be open. Otherwise, the passengers in the vehicle may get injured. The gate must wait for the vehicle to pass and the close.



```
SPEC AG !( vehicle.vehicleStatus = passing & railwayCrossing.railCrossingStatus=close )
```

Requirements

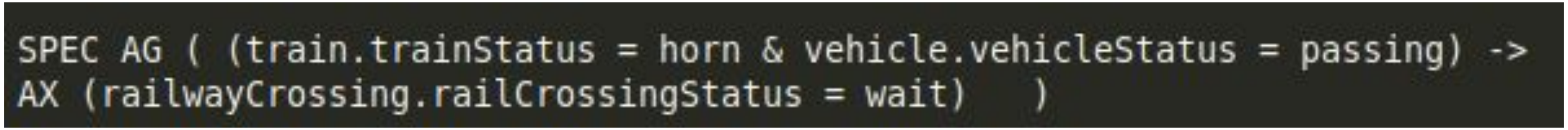
Starvation: We must ensure that if a vehicle is waiting at the crossing it will eventually pass. It must not wait indefinitely.



```
SPEC AG ( vehicle.vehicleStatus = wait -> EF vehicle.vehicleStatus = passing  )
```

Requirements

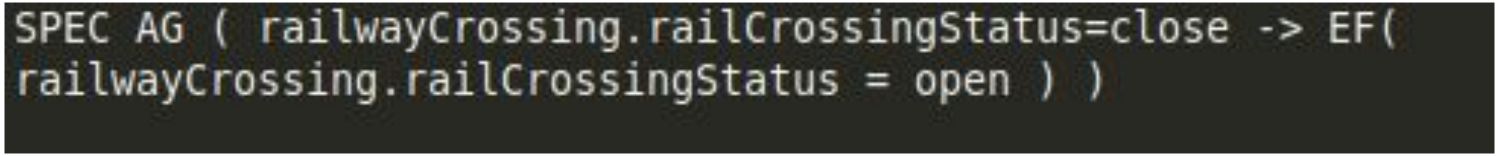
Safety of the vehicle when the railway crossing gate is going to close. So, whenever train sounds horn and vehicle are passing then in all nextstep railway crossing gate must wait for passing current vehicles.



```
SPEC AG ( (train.trainStatus = horn & vehicle.vehicleStatus = passing) ->
AX (railwayCrossing.railCrossingStatus = wait)   )
```

Requirements

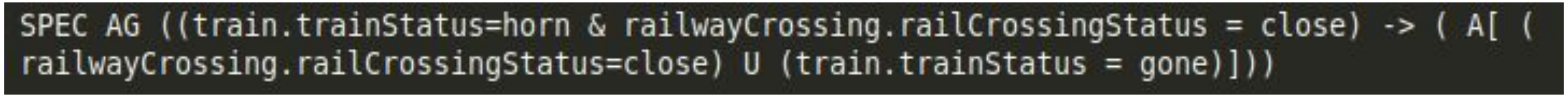
The gate must also open sometime in the future if it was closed.



```
SPEC AG ( railwayCrossing.railCrossingStatus=close -> EF(
railwayCrossing.railCrossingStatus = open ) )
```

Requirements

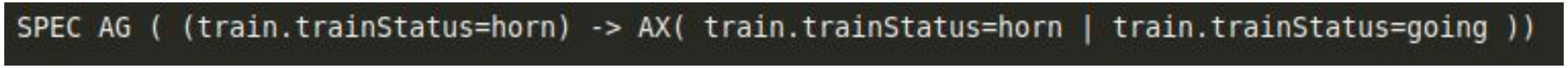
Whenever train sounds horn and railway crossing gate is closed then gate will remain closed until the train leaves.



```
SPEC AG ((train.trainStatus=horn & railwayCrossing.railCrossingStatus = close) -> ( A[ (
railwayCrossing.railCrossingStatus=close) U (train.trainStatus = gone)]))
```

Requirements

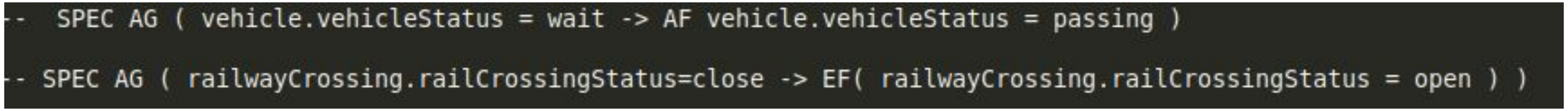
This is to ensure that there is no fake horn. The train will be on track after the horn.



```
SPEC AG ( (train.trainStatus=horn) -> AX( train.trainStatus=horn | train.trainStatus=going ))
```

Requirements

Below 2 rules are false because if the train comes continuously and the railway crossing gate will not open.



```
--  SPEC AG ( vehicle.vehicleStatus = wait -> AF vehicle.vehicleStatus = passing )

-- SPEC AG ( railwayCrossing.railCrossingStatus=close -> EF( railwayCrossing.railCrossingStatus = open ) )
```

Conclusion

1. We have designed a railway crossing gate controller.
2. We ensure the safety of passengers at all times. Here one assumption is made that the train can’t be controlled in any way. Specifications are used to verify the model, considering the safety issues.
3. This can lead to starvation in a very special case(If the train keeps on coming). Although this doesn’t happen in real life as two trains never run on the same track so close to each other.
4. We could improve this implementation by adding light for the train (a traffic light).

Thank You